ARMY RESEARCH LABORATORY

Automated Routing of Unmanned Aircraft Systems (UAS)

by Edward M. Measure, David Knapp, Terry Jameson,
and Andrew Butler

ARL-TR-4916 September 2009

Army Research Laboratory
White Sands Missile Range, NM 88002-5513

ARL-TR-4916 **September 2009**

Automated Routing of Unmanned Aircraft Systems (UAS)

Edward M. Measure, David Knapp, and Terry Jameson
Computational Information Sciences Directorate, ARL

Andrew Butler
Physical Science Laboratory, New Mexico State University

REPORT DOCUMENTATION PAGE						***Form Approved OMB No. 0704-0188***
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**						
1. REPORT DATE *(DD-MM-YYYY)* September 2009		**2. REPORT TYPE** Final				**3. DATES COVERED (From – To)** October 2006 to June 2009
4. TITLE AND SUBTITLE Automated Routing of Unmanned Aircraft Systems (UAS)						**5a. CONTRACT NUMBER**
						5b. GRANT NUMBER
						5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Edward M. Measure, David Knapp, Terry Jameson, Andrew Butler						**5d. PROJECT NUMBER**
						5e. TASK NUMBER
						5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory Information and Electronic Protection Division Computational Information Sciences Directorate (ATTN: RDRL-CIE-D, RDRL-CIE-M) White Sands Missile Range, NM 88002-5513						**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-4916
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)						**10. SPONSOR/MONITOR'S ACRONYM(S)**
						11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT Unmanned Aircraft Systems (UAS) have become a key component of US military power and are likely to have an increasing role in reconnaissance, surveillance, communication and combat. UAS operations are affected by weather and other environmental effects, but usually have less capability to see, react to, and endure adverse environments than manned aircraft. Weather effects thus become a crucial part of both operational planning and execution of UAS missions. The U.S. Army Research Laboratory (ARL) has devised a weather effects tactical decision aid, which uses systems performance parameters, a weather effects database, and observed and predicted meteorological (Met) parameters to plan routes through weather and other hazards to carry out missions with maximum effectiveness and minimal mission risk.						
15. SUBJECT TERMS UAS, algorithms, automated routing.						
16. SECURITY CLASSIFICATION OF:				**17. LIMITATION OF ABSTRACT** UU	**18. NUMBER OF PAGES** 46	**19a. NAME OF RESPONSIBLE PERSON** Dr. Edward Measure
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified				**19b. TELEPHONE NUMBER** *(Include area code)* (575) 678-3307

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures **v**

List of Tables **v**

Preface **vi**

1. Summary **1**

2. Background and Problem Statement **1**

2.1 The UAS Mission 1

2.2 Met Effects 2

2.3 Problem Statement 2

2.4 Approach to the Problem 3

3. Actionable Weather Intelligence **4**

3.1 Weather 4

3.2 Weather Impacts on Army Systems 4

4. Route Planning: Avoiding Adverse Weather Impacts **5**

4.1 Manual Routing 6

5. Automated Routing: Graphical Idealization **7**

5.1 Reduction of Path to a Graph 8

5.2 Cost 9

5.3 Computing Path Costs 9

6. Automated Routing: Searching the Graph **12**

6.1 Automating Path Search 12

6.2 Application to UAS Route Planning 13

6.3 Search Algorithms 13

6.3.1 Breadth-First 14

6.3.2 Dijkstra’s Search Algorithm 16

6.3.3 Best First Search Algorithm 20

6.3.4 The A* Algorithm 21

7. AWRT Examples and Screen Shots 25

8. Conclusions and Future Directions 27

9. References 28

Appendix A. Examples of the A* Algorithm in Action 29

Appendix B. A*Algorithm Code for Route Finding 35

List of Symbols, Abbreviations, and Acronyms 37

Distribution 38

List of Figures

Figure 1. Diagram of IWEDA based route planning. 7
Figure 2. One route from 1A to 3C. 10
Figure 3. Breadth-first algorithm. 15
Figure 4. Dijkstra’s algorithm pseudo code. 17
Figure 5. Flow chart of Dijkstra’s algorithm. 18
Figure 6. Dijkstra’s algorithm. 19
Figure 7. Best First Search algorithm. 20
Figure 8. The A* algorithm pseudo code. 22
Figure 9. The A* algorithm flow chart. 23
Figure 10. The A* algorithm. 24
Figure 11. User interface for T-IWEDA route planning feature. 25
Figure 12. Straight line path for UAS. 26
Figure 13. Path optimized for weather effects. 26
Figure A-1. A simple impenetrable obstacle. 29
Figure A-2. The initial stages of the A* algorithm search. 30
Figure A-3. Target found and path is being defined. 31
Figure A-4. The shortest (cheapest) path. 32
Figure A-5. Tackling a complex path. 32
Figure A-6. The same search at an intermediate point. 33
Figure A-7. Defining the path. 33
Figure A-8. The shortest path. 34

List of Tables

Table 1. Cost assignment factors. 12

Preface

Unmanned systems have become a ubiquitous component of the American military. Such systems can conduct a wide variety of tasks that minimize the risk of casualties, including counter mine and demolition operations, force projection, and above all, reconnaissance, surveillance, and intelligence operations. To date, the largest share of such operations, and the best payoff for investment, has come with Unmanned Aircraft Systems (UAS). UAS now exist in a great variety of sizes and capabilities.

Like other aircraft, UAS are vulnerable to weather. Many of them are small with low wing loadings. Usually, they are less robustly designed than typical manned aircraft, and some are intended to operate at low altitudes. These factors tend to make them more vulnerable to the weather than their larger and far more expensive manned counterparts. These vulnerabilities mean that there is a combat multiplier for the mission planner who takes into account these vulnerabilities and plans accordingly.

Therefore, the U.S. Army Research Laboratory’s (ARL) Battlefield Environment Division (BED) has developed tools to facilitate that planning and maximize the efficient use of the UAS. This report describes one such tool, which is the Aviation Weather Routing Tool (AWRT) automated route planner.

1. Summary

Unmanned Aircraft Systems (UAS) are affected by the same atmospheric phenomena as manned aircraft and in many ways are more vulnerable to them. Planning and carrying out UAS missions in the most effective manner requires that the planning process properly takes into account these effects.

The U.S. Army Research Laboratory (ARL) has developed an Integrated Weather Effects Decision Aid (IWEDA) that uses four-dimensional (space plus time) meteorological (Met) information, a database of weather impacts on Army systems, and expert systems technology to produce time and location specific actionable weather intelligence for a large variety of Army systems.

In this technical report, we describe the way we have used IWEDA to develop an automated weather routing tool that finds best paths for UAS vehicles operating in the presence of weather. This tool uses the weather impacts information to assign costs for each segment of the mission and uses a computer-based path-finding tool, the A-star (A*) algorithm, to find the optimal route for the vehicle.

The routing tool has been developed and demonstrated. Future enhancements being developed include integration of in-flight weather data, and incorporation of methods for use for the full spectrum of UAS missions.

2. Background and Problem Statement

UAS are affected by the same atmospheric phenomena as manned aircraft, including (but not exclusively) cross and tail winds at takeoff and landing, turbulence, icing, visibility, clouds, precipitation, and severe weather en route. Relevant weather information is needed for flight planning, and current methods for providing that information and planning are inadequate. Our automated routing system can provide that information in a form that will take weather and other impacts into account when planning UAS routing. This report examines the basic elements of our method for providing automated weather and other routing information for UAS operations.

2.1 The UAS Mission

UAS are capable of locating and recognizing major enemy forces, moving vehicles, weapons systems, and other targets that contrast with their surroundings. In addition, UAS are capable of locating and confirming the position of friendly forces, presence of noncombatant civilians, and so forth. Current Army UAS missions include:

- Reconnaissance.
- Surveillance.
- Security.
- Manned-Unmanned Teaming.
- Communications Relay (HQDA [website accessed August 2009]).

UAS have been used extensively by all branches of the Department of Defense (DoD) during the past decade in battlefield theaters of operation for a variety of purposes (weapons delivery, reconnaissance, psyop pamphlet drops, etc.). More recently, UAS have been employed in border patrol and other homeland defense missions. Because of their tremendous versatility, the use of UAS continues to rapidly expand.

2.2 Met Effects

For most Army UAS missions, these remotely-operated vehicles are typically small and light-weight (often smaller than common single-engine passenger aircraft) and are quite vulnerable to various Met hazards. These vulnerabilities result in part from the size, weight, power systems and relatively simple designs used for systems that are intended to be lighter, cheaper, and more expendable than manned systems. Potential sources of vulnerability noted in the previous reference include turbulence, clouds, precipitation, wind, visibility, temperature, and illumination. These Met phenomena manifest themselves in effects on aircraft survivability, sensor performance, aircraft performance and aircraft range.

Icing conditions, including carburetor icing effects, runway crosswind components that exceed aircraft control capabilities, and extreme temperatures that negatively affect the aircraft’s performance are adverse to UAS operations and must be considered. Met effects need to be accounted for when determining UAS sensor performance, aircraft flight control, data transfer communications, and risk of detection of the vehicle en route and over target areas by unfriendly ground and air forces. Precipitation can erode and possibly destroy the small, rapidly spinning propellers of some UAS.

2.3 Problem Statement

Operationally today, mission planning and flight route weather information along the planned UAS routes have been conveyed to the planners and operators via a DoD standard pilot weather briefing form. This form presents information primarily in text format or with simple map sketches covering broad flying regions and across extended timeframes. It is left to the UAS mission planners and operators to infer the specific Met conditions along the intended route at the time the aircraft arrives at particular route waypoints and the impact of those conditions on the mission. Deducing weather impacts is beyond the training of typical Unmanned Air Vehicle (UAV) mission commanders, and the manual planning of routes is time-consuming and complex, even for Met experts. The current inability to provide accurate, timely, and detailed Met information, potential system impacts (on the aircraft itself and its on-board sensors), and alternative routing options to UAS operators represents a serious lack of capability that reduces mission success rates.

The incorporation of state-of-the-art Met information into UAS operations lags far behind the aircraft and sensor technologies currently being employed and those planned for future fielding. Some preliminary UAS Met Tactical Decision Aid (TDA) development work was previously accomplished within the IWEDA database hosted on the fielded Integrated Meteorological System (IMETS). The IWEDA weather impacts rules pertaining to UAS operations are very preliminary and generic. The current suite of IMETS graphics products, which can depict UAS TDA information, is not well-suited to operator needs. Therefore, a lot of work needed to be done to develop user-tailored UAS en route weather conditions and routing options to avoid adverse conditions. Our work here addressed the need to develop and deliver TDA technology to include more timely and accurate Met information and impacts to UAS mission planners and operators in more useful formats addressing a variety of operational scenarios.

2.4 Approach to the Problem

The technology of automated route planning has reached a stage of considerable maturity, due in part to the needs of computing, the internet, and computer games. Routing is needed on the internet to plan the routes by which messages are sent from one location to another. Computer games use automated routing techniques to move computer characters around the screen, while the computing process itself may use automated routing techniques to plan the execution of interdependent bits of program code.

Applying these techniques to the aircraft routing problem requires at least the following components:

- A database of current and projected basic Met parameters (temperature, pressure, a measure of humidity and winds)
- A database of derived Met parameters (icing, turbulence, precipitation rates, cloud volumes, etc.)
- A method for computing or otherwise finding weather impacts on the vehicle in question
- A method for generating three-dimensional paths between the takeoff point and mission areas
- A method for comparative evaluation of the potential paths

The first three of the five components were adapted from the IWEDA, while the latter two were developed in the current effort. This report discusses the status and prospects for each of these components with primary focus on those elements that were developed here.

3. Actionable Weather Intelligence

3.1 Weather

Our information about the current state of the atmosphere can be encoded as a three-dimensional table with each element of the table corresponding to a three-dimensional sub-volume of the atmosphere. The information stored in that element describes the atmospheric state in that sub-volume: temperature, humidity, wind speed and direction, turbulence, the presence or absence of cloud or fog, precipitation and so on. Because the atmosphere has structure most of the way down to the molecular level, such values are necessarily averages. Such tables are the product of measurements, analysis, and atmospheric prognostic models.

A tremendous world-wide effort goes into the measurement of the atmosphere. Thousands of weather stations, hundreds of twice-daily balloon soundings, hundreds of weather radars and the globe-spanning satellites are continually monitoring the state of the atmosphere. The weather analysis is constructed from all of these measurements, and it is interpolating where measurements are missing or in conflict, and constructing averages for regularly spaced volumes.

Our measurements of the atmosphere would mainly be of historical interest if it were not for one all-important fact: we know the laws of atmospheric behavior, encoded in the Navier-Stokes equations it obeys. Except for two major problems, those equations allow us to predict the future state of the atmosphere from its present state. The first problem is in order to predict the future precisely, we need to know the present precisely, and the second problem is that those equations are very difficult to solve and cannot be solved exactly except in very artificial circumstances. The equations can be solved approximately using our necessarily approximate measurements as input and with the help of very powerful computing systems. The programs that construct those approximate solutions are called Numerical Weather Prediction (NWP) models. NWP models output a grid of values of atmospheric parameters. The advance of technology has increased the accuracy and density of our measurements. Meanwhile, computing power has increased even more rapidly, resulting in increasing accuracy of weather predictions.

3.2 Weather Impacts on Army Systems

An important project of the ARL BED has been the effort to extract more utility from the weather data describing the atmospheric state by supplementing that data with information about weather impacts. Determination of weather impacts requires information about the requirements and vulnerabilities of Army systems, as well as of the weather state.

Because there are many Army systems and many kinds of ways that weather can potentially impact them, the task of evaluating all the impacts on any given operation can become very complex. Such a complex and data-driven process is a good candidate for automation, or at least partial automation. Developing such automation has been a major focus of ARL BED for some years.

ARL has implemented a key component of such automation by the compilation of a large and expanding database of weather impacts on Army and other DoD systems. For a large number of systems in the inventory, including Soldier systems, the effects of various values of weather parameters have been determined and their impact on the systems encoded in a specialized database for use by the IWEDA, now being transitioned to the Tri-Service Integrated Weather Effects Decision Aid (T-IWEDA). Currently, these effects are encoded as “red” for severe or catastrophic system degradation, “amber” for marginal operational conditions; and “green” for minor effect or no effect.

In the T-IWEDA, these rules are combined with information about the existing weather state as a function of location to determine a map of areas where systems or operations will be adversely affected by weather. The T-IWEDA is implemented via an expert system, which computes locations where systems are affected and displays the output on two-dimensional map display, which shows conditions as “red”, “amber”, or “green”. To use T-IWEDA, the operator selects an operational domain, the source of atmospheric data, and a system or suite of systems to be considered. The T-IWEDA then evaluates the weather impacts for all the systems considered and displays a map of the red, amber, and green regions for planning and operational use.

4. Route Planning: Avoiding Adverse Weather Impacts

ARL’s objective was to produce an automated routing system that could consider weather impacts on UAS and find paths that avoided the most dangerous weather hazards. What is needed in order to plan routes to avoid the adverse impacts of weather? The requirements are:

- Information about the weather.
- The movement requirements of the mission.
- The systems required for the mission.
- Methods needed to evaluate the potential impacts of the weather on the required systems, impacts that will usually be a function of time and location.
- A means to plan routes for the mission systems that avoid the worst impacts of the weather.

The first requirement is supplied by our weather nowcast databases, and the second and third requirements need to be provided by the operational planners. The crucial role of evaluating the weather impacts and their variation with time and location is played by the T-IWEDA system discussed above. Mission planners and UAS commanders currently perform the route planning task itself.

4.1 Manual Routing

In order to perform their task, mission planners and commanders rely on tools, such as weather maps and text messages. These tools are often supplemented by a single vertical cross-sectional analysis of the weather in the general area of the planned route. Satellite imagery and numerical weather prediction models can provide atmospheric analysis at considerably higher resolution, but it is difficult and complex to manually convert this information into weather impacts on systems. Finally, the T-IWEDA system organizes much of this information in a way that permits mission planners to see the weather hazards of potential routes by drawing them on an electronic map.

There are still difficulties that remain. If missions can operate at multiple altitudes, the task is complicated by the necessity to sort out effects by altitude. If, for example, the weather hazards only exist at a few levels, the map will nonetheless display a hazard for this block, and discovering the clear path requires drilling down to check each altitude for every potential path being considered. This remains a time consuming and complex task, with ample opportunity for error.

A manual version of flight routing software has already been incorporated into the T-IWEDA. To use it, the operator chooses the aircraft type; uses the mouse to select launch, target, and recovery points; and uses the keyboard to input flight altitudes at those points. The T-IWEDA system plots straight lines between these points, notes the grid cells passed through or near, and marks each of those cells red, amber, or green on the map, see figure 1 below. In figure 1, the block labeled "Route Planning" can be either manual or automated.

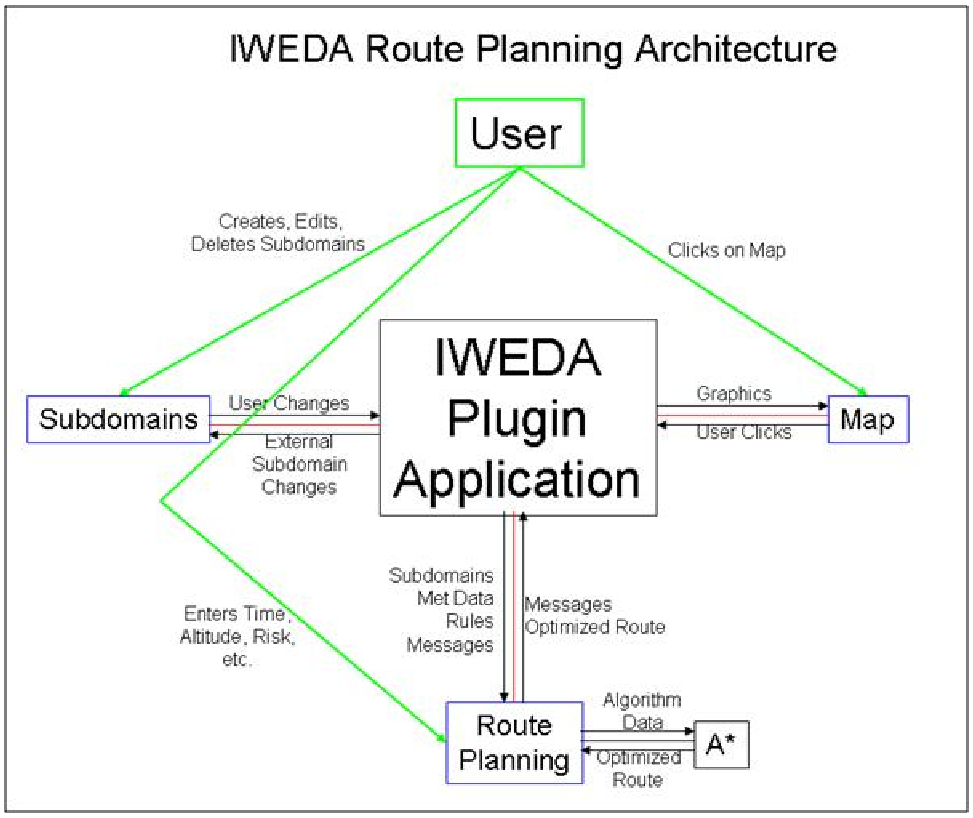

IWEDA Route Planning Architecture
User
Creates, Edits, Deletes Subdomains
Clicks on Map
IWEDA Plugin Application
Subdomains
User Changes
External Subdomain Changes
Graphics
User Clicks
Map
Subdomains Met Data Rules Messages
Messages Optimized Route
Enters Time, Altitude, Risk, etc.
Route Planning
Algorithm Data
Optimized Route
A*

Figure 1. Diagram of IWEDA based route planning.

If the mission is unfavorably impacted by weather effects, the operator may try choosing a different path or a different flight altitude, or if the mission allows it, a different launch time. It is this latter portion of route planning that can prove time and labor intensive, and that is the part our effort proposes to automate.

The difficulties and limitations of manual routing led us to consider the potential for automated routing by means of computer algorithms. Such routing has considerable potential for avoiding adverse weather impacts and also has the potential to be generalized to consider impacts beyond weather. There are several possible approaches to the path-finding, but all are based on using some variation of a cost function. Section 5 discusses some options for automated path-finding or routing.

5. Automated Routing: Graphical Idealization

For automated routing, much of the same core technology is used for determining the impacts, including the Met database, the T-IWEDA rule database for the systems considered, and the computed three- or four-dimensional database of impacts (the fourth dimension being time).

Additional mechanisms are needed for evaluating paths and for selecting alternate paths. In particular, we will need to simplify the search space of possible paths and, additionally, to attach information that will allow the computer to judge which paths are better.

5.1 Reduction of Path to a Graph

There are infinitely many paths between any two points in three-dimensional space, so how can we compare them and pick one? It's usual to avoid this difficulty by translating the problem to a simpler one by discretizing the relevant portion of space into a finite number of volume elements. Since we only have Met information for the volume elements of the weather grid, that approach is natural for our problem, and it's the one we used.

Once space is reduced to discrete blocks, the geometrical relation of the elements of space is reduced to a graph by letting the nodes of the graph represent the individual volume elements and the links between the nodes correspond to the connections between adjacent volume elements. This scheme captures the geometric fact that you can't travel from one volume in space to a distant volume without passing through a linked sequence of volume elements between them. To give a concrete example, imagine that A, B, C, and D are nodes and that A is linked to B and C, B is linked to C, and C is linked to D. In that case, A-C-D and A-B-C-D are paths from A to D but A-D and A-B-D are not since neither A nor B is linked to D. In three dimensions, the linking is more complicated, since each rectangular parallelepiped volume element shares faces with six other volume elements, edges with 12 more, and a corner with yet another eight.

For problems like routing over computer or other telecommunications networks this kind of reduction to a graph is extremely natural since their topology closely resembles the mathematical graph. The idealization is not much greater for routing over effectively one-dimensional connections such as highways or railroads, but idealization is clearly involved when a two-, three-, or more dimensional space is reduced to nodes and links.

Once the search space is reduced to a graph, any given volume contains a finite number of non-self-intersecting paths. Each path consists of a linked series of adjacent nodes, and costs can be assigned to each link between nodes, or to both nodes and links. It is important for the algorithms that we will use that all of those costs be positive, but this is not a serious limitation for our purposes. The cost of the path then becomes a sum or other accumulation of the costs for each segment traversed.

Because we wish indirect routes to be considered, it should also include some surrounding space. We can limit the size of the search space, though, by excluding regions that are forbidden. Examples would be cells containing impassible obstacles like mountains, cells above the UAS flight ceiling, and regions dedicated to incompatible operational uses.

The path-finding problem has now become a graph theory problem: find a sequence of linked nodes connecting the starting point to the destination point. We don't want just any path either;

we want the best path, or at least, an acceptable cost path. We have mainly been concerned with costs due to adverse weather, so our cost function is determined from the weather. Other impacts on the system in question can be computed similarly on the basis of other mission risks, but we have not done that here.

5.2 Cost

How does the computer evaluate which path is "best" or even "good enough"? The key idea is the association of a computable cost function with each path, and searching the alternatives for the best or lowest cost path. In principle, such a cost function can reflect not only weather effects but also other constraints. There might be restricted flight corridors reserved for other aircraft operations, or paths that would take the UAS too close to anti-aircraft fire, or a need to prevent acoustic detection until the latest possible moment, as well as the very fundamental requirement that the UAS should not unintentionally fly into any mountains or other obstacles. Our initial implementation is intended to reflect only the basics plus weather—don't fly into the ground, don't run out of fuel, and minimize adverse weather impacts, but we will show how the method can readily be extended.

Implementation requires methods for computing the path cost and ways to deal with the potential infinitude of possible paths. Since we are trying to avoid adverse weather here, system specific costs are associated with adverse or potentially adverse weather. Our weather data, and consequently our weather impacts, have finite resolution and are specified for elements of a grid in space and time. At a given time step, the weather and weather impacts are represented by single values for each parameter, and each such value is considered to be an average over that grid volume element.

5.3 Computing Path Costs

Traversal of each grid element is assumed to have a cost, or risk, which is a function of the weather in that grid element. A path between locations A and B is considered to be a list of adjacent grid elements beginning at A and ending at B, and the cost of that path is an accumulation of those costs over the elements of that path. Figure 2 is an example of a very short two-dimensional path of just three segments, which visits four nodes.

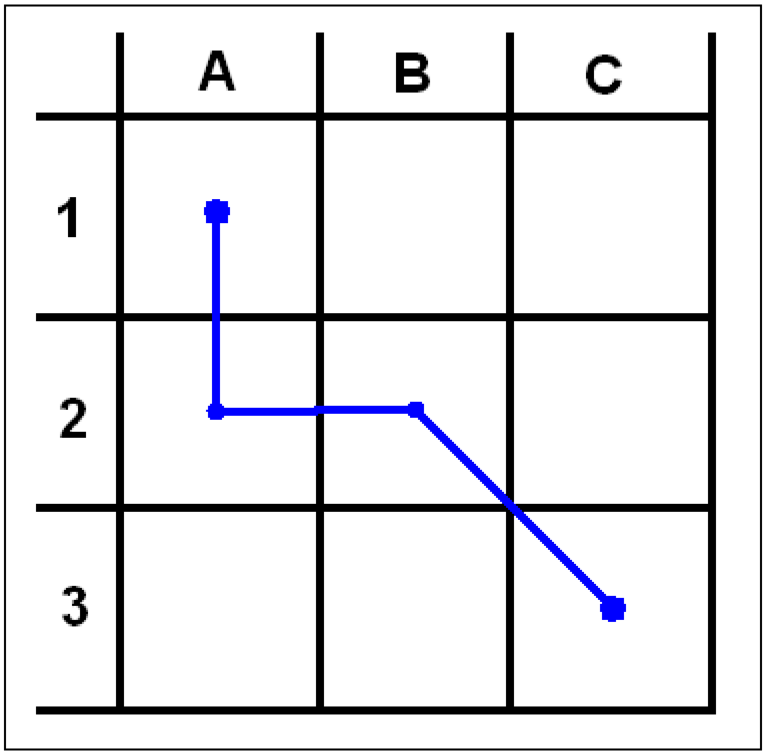


Figure 2. One route from 1A to 3C.

In the example above, the cost of the path shown between A1 and C3, or $C_{A1,C3}$ can be expressed as $C_{A1} + C_{A1,A2} + C_{A2} + C_{A2,B2} + C_{B2} + C_{B2,C3} + C_{C3}$ where $C_{A1,\ A2}$ is the cost to travel from point A1 to point A2, C_{A2} is the cost to exist on point A2, and so on. A common cost associated with travel is distance or time, and a common cost associated with existing on a point is whether there is an obstruction there. Further discussion and examples can be found in Patel (website accessed February 25, 2006). Detailed discussion, and the original derivation of the principle methods used here can be found in Hart, et al., 1968 and Rensselaer Polytechnic Institute (website accessed 1999).

In order to assign costs to the relevant elements of potential paths for the UAS mission, the Met parameters database and the weather effects database must include the mission launch, recovery, and target areas, as well as the intervening space. The T-IWEDA rules are used to compute costs per segment and cost per node.

Most of the weather impacts portion of the cost is computed from the weather impacts thresholds in T-IWEDA. The T-IWEDA rules specify values of weather parameters, such as turbulence, that present a severe risk “red”, moderate risk “yellow”, or little or no risk “green”. In T-IWEDA, the effects are computed as Boolean values— it is either true or false that a parameter value exceeds the “red” or “yellow” threshold found in a rule. The cost function, however, is computed on a more continuous basis. Those values are used to compute costs for path segments passing through a region with those parameter values. Since the costs are numbers, the costs have finer resolution than the “red”, “yellow”, “green” T-IWEDA status. The cost is determined by comparison of the parameter value with the rule thresholds. Thus, values that just top the “yellow” threshold and might pose a slight risk are getting lower cost s than values that are close to the “red” threshold.

An example cost calculation might concern a vehicle with a sensor package that was vulnerable to cloud (because the lenses fog). Suppose that the T-IWEDA thresholds were “yellow” for a probability of greater than 30% that the given volume would have too much cloud and “red” when that probability was greater than 60%. One possible assignment of cost would be C=(p-20)^2/100 for p > 20 and 0 otherwise. Under this system, the “green” value 28% translates to 0.64, the barely “yellow” value p = 32% to C = 1.44, the seriously “yellow” of p=60% becomes C=16.0, and “red” 90% becomes C=(90-20)^2/100 =49. In practice, we more usually made cost a linear function of the threshold violation.

Weather conditions that are not necessarily severe can also play a role. One important consideration in mission planning is fuel consumption. Fuel consumption is only a major consideration when there is a danger of running out. In that case, it threatens loss of vehicle or failure of mission and a potentially catastrophic cost. Consequently, it is useful to have a fuel cost function that keeps track of how much fuel is likely to be used on a potential path.

Fuel consumption depends on a number of things, including speed and altitude, but especially on the wind. Small UAVs in particular are rather slow flyers, so a strong cross wind or head wind can greatly lengthen a mission and increase fuel consumption. We consider this cost computation in some detail below.

Let the fuel consumption rate be f = f (air speed, rate of climb, etc.). The fuel consumed in moving through the sub-volume is f*t, where t is the time taken to traverse the sub-volume. If the UAS travels a distance **d** in traversing a grid sub-volume, then $\mathbf{d} = \mathbf{v}_g * t$, where $\mathbf{v}_g$ is the ground speed, so $t = |\mathbf{d}| / |\mathbf{v}_g|$. Here, and subsequently, bold will be used for vector variables. The notation $|\mathbf{d}|$ refers to the length of the vector **d**.

In equation 1, the ground velocity is related to the air velocity and the wind velocity by:

$$\mathbf{v}_g = \mathbf{v}_a + \mathbf{V} \qquad (1)$$

where $\mathbf{v}_a$=the velocity relative to the air of the UAS, and **V**= the wind speed in the grid sub-volume in question.

Thus, in equation 2, the fuel consumption in the grid cell is given by:

$$C_f = \frac{|\mathbf{d}|}{|\mathbf{v}_a + V|} \qquad (2)$$

Equation 2 is just the cost for one grid volume's fuel use. In order to calculate the cost for a whole path, it is necessary to add up the cost for all the elements of the path. For fuel, that is a simple addition. In equation 3, once a path or path segment is identified, the fuel costs for each sub-element are computed, and the total cost is added up:

$$C_f = \sum_{i=1}^{N} C_f(i) \tag{3}$$

where 'i'=the path segment, and C_f (i) =the cost associated with that segment.

Fuel cost is a good example of a go or no-go cost. Since fuel consumption is usually only a problem when there is risk of running out, it's probably sensible to have the fuel cost be a separate component of the total cost with a threshold. That is, its contribution to the total cost would be small until a critical threshold was reached (in danger of running out of fuel) and then rapidly increase to a large value.

Table 1. Cost assignment factors.

1.	Impacts of weather on T-IWEDA rules for an asset are included in calculations and priority is given to unfavorable impacts over marginal impacts.
2.	Altitude restrictions may be accounted for by restricting the search space.
3.	Flight time is accounted for using ground distance traveled, as well as wind information to calculate airspeed. A constant (user-defined) air speed is assumed. Due to this fact, fuel consumption may be determined accurately from the generated route's flight time.
4.	The degree of acceptable risk that the user enters is used to determine the relative weight assigned to weather impacts as opposed to flight time. So a "careful" path will take more time to avoid weather impacts, while a more "risky" path will attempt to save time by cutting through small patches of harsher weather.
5.	The flight time for an asset is accounted for by choosing weather forecast data times closest to the asset's current time, which is adjusted granularly for each new cell that the asset passes through.

6. Automated Routing: Searching the Graph

6.1 Automating Path Search

It remains to be explained how the computer finds a suitable path from one location to another in the graph. The basic idea is for the computer to start at one of the endpoints and consider first the paths to adjacent nodes, then to the nodes adjacent to those, and so on until it encounters the target nodes. Meanwhile, it has been storing information about the partial paths it has constructed, which permits it to choose which one is of lowest cost. As we will see, there are many variations to this idea, and each is with its advantages and disadvantages.

In the most general graph, it might be necessary to consider every possible path between takeoff point and destination in order to find the least cost path. Ours has some simplifying features. In particular, costs are always positive. For example, you can't decrease cost by going around a loop. Also, we will assume that costs don't depend on the order in which the nodes are traversed. Finally and very importantly, we will keep in mind that our graph is a representation of an underlying metric structure (the ordinary three-dimensional Euclidean distance), and that information makes possible *informed search*, which often leads to a more efficient search.

The obvious idea behind informed search is that if you know in which direction the target is, it might be a good idea to look first in that direction. That notion is not sufficient for a computer algorithm, since the straight line path may lead into an obstacle or a *cul de sac*. One way to capture the information that makes possible informed search is to make use of a *heuristic.* In our case, the heuristic used is a number associated with each node that is proportional to the Euclidean distance from that node to the target or destination node.

Thus, we have two pieces of information upon which to work: the cost function g(x), which measures the cost of getting to a given node from the start node, and the heuristic function h(x), which depends only on the distance from the given node to the target node. The cost function will, of course, depend not only on the given node but on the path taken to get to that node. It is a very important simplification that we can choose a strategy that makes it possible to assign this cost uniquely without searching all possible paths.

6.2 Application to UAS Route Planning

Several complexities come into play when we attempt to apply the technique discussed above to UAS route planning. Perhaps most obviously, we are talking about travel through three-dimensional space. This is not an obstacle in any fundamental sense, but it does mean that the number of search space cells to be explored grows with distance at a more rapid rate (x^3 versus x^2). In consequence, the advantage for an algorithm using a distance-to-target heuristic tends to be more pronounced.

6.3 Search Algorithms

Our type of search is special in several ways. First, we assume a known destination point and a known starting point. Second, we assume that the costs of each link are known before the search begins. In addition, we will assume that all costs are positive. Most importantly, we do have a useful heuristic, since we can compute the Euclidean distance from any given node to the destination node. Finally, we assume that the cost of a link is independent of the order in which links are traversed.

It may help visualize how our algorithms work by considering the task of finding the shortest path between points. For that case, the cost function is just the distance traversed between the points. A straight line is the shortest path if there are no obstacles, but obstacles mean that the distances travelled in alternate routes around them need to be compared. Computerized search of a graph requires a certain amount of bookkeeping. We assume that we start with a knowledge of what is connected to what (for our case, volumes in space are connected to each adjacent volume) and what cost is associated with such connection. The algorithms we will consider each involve a systematic search outward from the start point in an attempt to reach the destination.

In particular, we will need to keep track of what parts of potential paths we have looked at and don't have to be further examined. For this, we will need a couple of lists of nodes, each initially empty, which we will refer to as the "open list" and the "closed list". We also need the concept of the parent node, which will be key to organizing our partial paths. The start node will be parentless, but every other node which reaches the closed list will have a unique parent. A chain of parent-child relationships to the start node will connect the node. This permits every node on the closed list to define a path, terminating on that node and, defined by the chain of parent-child relationships, originating on the start node and terminating on the node in question.

It's useful to note that the parent-child relationships define a specific type of directed graph, a tree graph, which is a subgraph of the original graph. This graph has all links, except the parent-child links, erased and the parent-child links are made one-directional from parent to child. The virtue of the tree graph is that there is a unique path from each "bud" of the tree back to its "trunk". When one of these buds reaches the destination node and the desired path has been found, it is just the reverse of child-parent path from bud to trunk. See, for example, figures 3, 6, and 7, which represent the parent-child relationship by arrows, each of which should be thought of as pointing to the current node (square) from the direction of the parent node (square). The chains of green squares in the figures represent the path found. Note that each green square is the parent of the next one and the child of the previous one.

Starting in section 6.3.1, we consider four different search algorithms, which could be employed and finishing with the one we actually used. The point of considering the others is that their strengths and weaknesses illustrate the nature of the problem and its potential pitfalls.

6.3.1 Breadth-First

Probably the most straightforward search algorithm is breadth-first. The strategy here is to expand outward from the start node. Wikipedia has an excellent but slightly incomplete description:

1. *Put the root node on the queue.*
2. *Pull a node from the beginning of the queue and examine it.*

   a. If the searched element is found in this node, quit the search and return a result

   b. Otherwise push all the (so-far-unexamined) successors (the direct child nodes) of this node into the end of the queue, if there are any.
3. *If the queue is empty, every node on the graph has been examined-quit the search and return "not found"*
4. *Repeat from Step.* (Wikipedia, [website accessed August 8, 2009]).

The description above leaves out an important detail. In step 2b., when the unexamined successor nodes are pushed on to the queue, the current node is marked as its "parent". Thus each node that has entered the queue has a unique parent. When the target element is encountered, there is, as noted above, a unique backtraced path from parent to parent all the way back to the start element. The path thus found, when reversed, is the sought-for path from start node to target.

In terms of our lists, step 1 consists of adding the start node to the open list, and step 2a. checks each member of the open list (the "queue") to see if we have reached the goal, and, if not, step 2b. moves the current node to the closed list and all nodes adjacent to closed nodes to the open list (see figure 3).

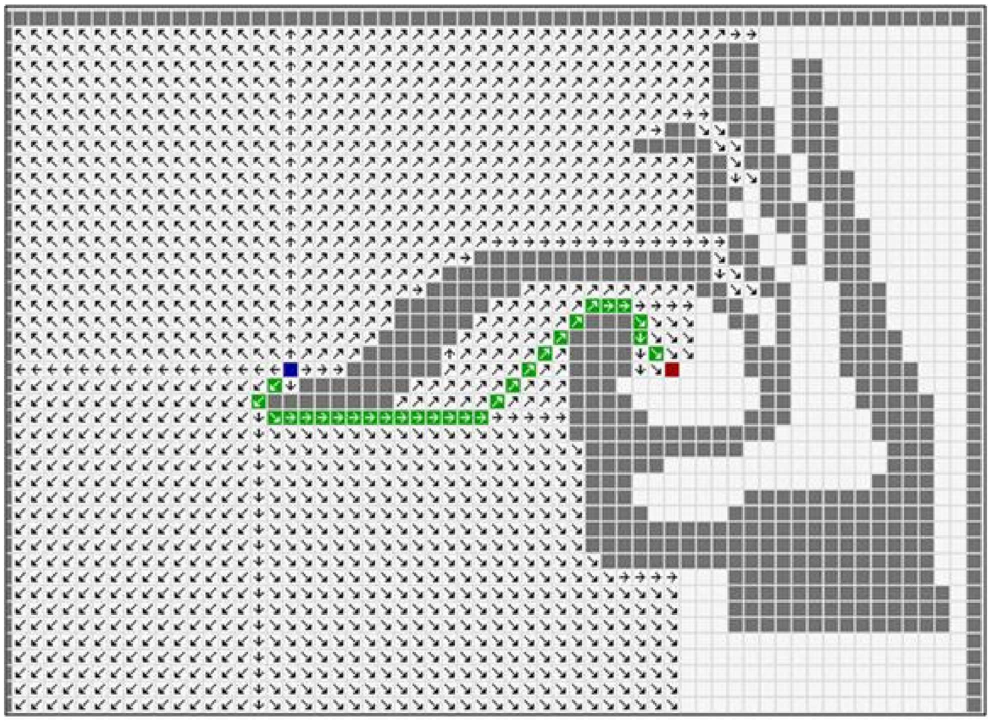


Figure 3. Breadth-first algorithm.

The breadth-first algorithm finds a path, but is often quite inefficient. In general, the path found will be minimal in number of links but not in cost. “Blue” is the starting node, “red” is the destination node, and gray squares are blocked. Arrows in squares point to the square (node) and from its parent square. Squares with arrows were tested, and those without were not. The green squares form the path found by the algorithm. Figure 3 was produced using an applet written by Baur (website accessed 2008).

The breadth-first algorithm will find a path, if it exists, (see figure 3) but that path is not guaranteed to be the lowest cost. No account of cost at all was taken in the algorithm, but the path found is minimal in the sense that it contains the least possible number of links. Since it successively searches all accessible links until it reaches the target, it is likely to be inefficient.

6.3.2 Dijkstra’s Search Algorithm

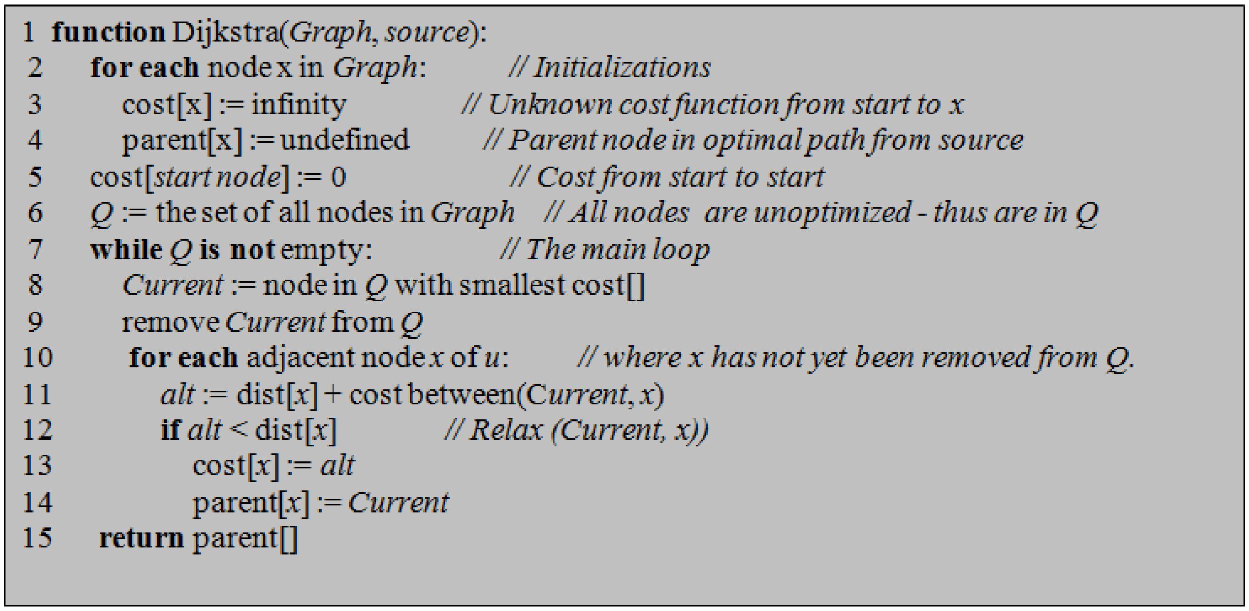
Dijkstra’s algorithm has many points of similarity to the breadth-first algorithm, but one crucial difference – it finds a lowest cost path. In the breadth-first algorithm, the nodes on the open list are all the same link number from the source, and since every open node is explored before adding new nodes to the open list, the order in which they are explored is unimportant. Each node of a “shell” of nodes of link number n is explored before any node of link number n+1. Since cost is not counted, any adjacent node in the current shell is an equally suitable parent for a node in the next shell. This makes possible a very simple algorithm for assigning parenthood. Considering the nodes in the current shell one at a time, it may be assigned as parent to every adjacent shell, which does not yet have a parent.

This simple scheme is not adequate for Dijkstra’s algorithm, since a given open node may be reachable from several different already-explored nodes, and the costs will not always be the same for each. Moreover, the lowest cost path to a given node might not come from the previous shell. A path of link distance n + m might turn out to be cheaper than any path of link distance m. Consequently, a certain amount of technology is necessary to unambiguously assign costs in a step by step fashion.

The idea is whenever a current node is added to the closed list, any adjacent nodes not already open or closed are added to the open list. The cost g(x) of each such adjacent node is compared with the g’(x), the cost of the current node plus, the link cost from the current node, and if g’(x) is less than g(x), g(x) is reset to the value of g’(x), and the current node is made the parent of that adjacent node. Cost and parenthood are still tentative at this point, since it is possible that a lower cost path, possibly with more steps, exists, which does not pass through the current parent. The current node is then moved to the closed list, and the open list is searched for lowest cost node.

This is the same procedure that led to the tentative cost, but now that the previous node has been moved to the closed list, it can be seen to be the actual cost (and similarly the procedure that formerly gave the tentative parent now gives the actual parent). The only other possibilities are that there is a lower cost path, either directly from some other closed node or passing through some other closed node to an open node and thence to the node in question. The first possibility can be excluded since if there were another closed node adjacent from which a lower cost path could have been found it would have been found during the addition of that node's adjacent nodes. The same logic dictates that no other open node could be reachable more cheaply than the node being examined since the cheapest path to it would either have to pass through the current node or one of the other closed nodes, and costs are assumed to be always additive and positive. Thus, the cheapest open node is added to the closed list, and then it becomes the new current node, and the process continues.

Because a very similar logic is used in the A* algorithm, the algorithm that we decided to use, we provide pseudo code for Dijkstra's algorithm below. The following pseudo code for Dijkstra's algorithm is adapted from Wikipedia (website accessed August 18, 2009). The pseudo code given is simplified in that it computes the distance from the start node to every node in the graph (see figure 4).



```
function Dijkstra(Graph, source):
    for each node x in Graph:          // Initializations
        cost[x] := infinity            // Unknown cost function from start to x
        parent[x] := undefined         // Parent node in optimal path from source
    cost[start node] := 0              // Cost from start to start
    Q := the set of all nodes in Graph // All nodes are unoptimized - thus are in Q
    while Q is not empty:              // The main loop
        Current := node in Q with smallest cost[]
        remove Current from Q
        for each adjacent node x of u:     // where x has not yet been removed from Q.
            alt := dist[x] + cost between(Current, x)
            if alt < dist[x]           // Relax (Current, x))
                cost[x] := alt
                parent[x] := Current
    return parent[]
```

Figure 4. Dijkstra's algorithm pseudo code.

Figure 5 below illustrates the flow chart for Dijkstra's algorithm.

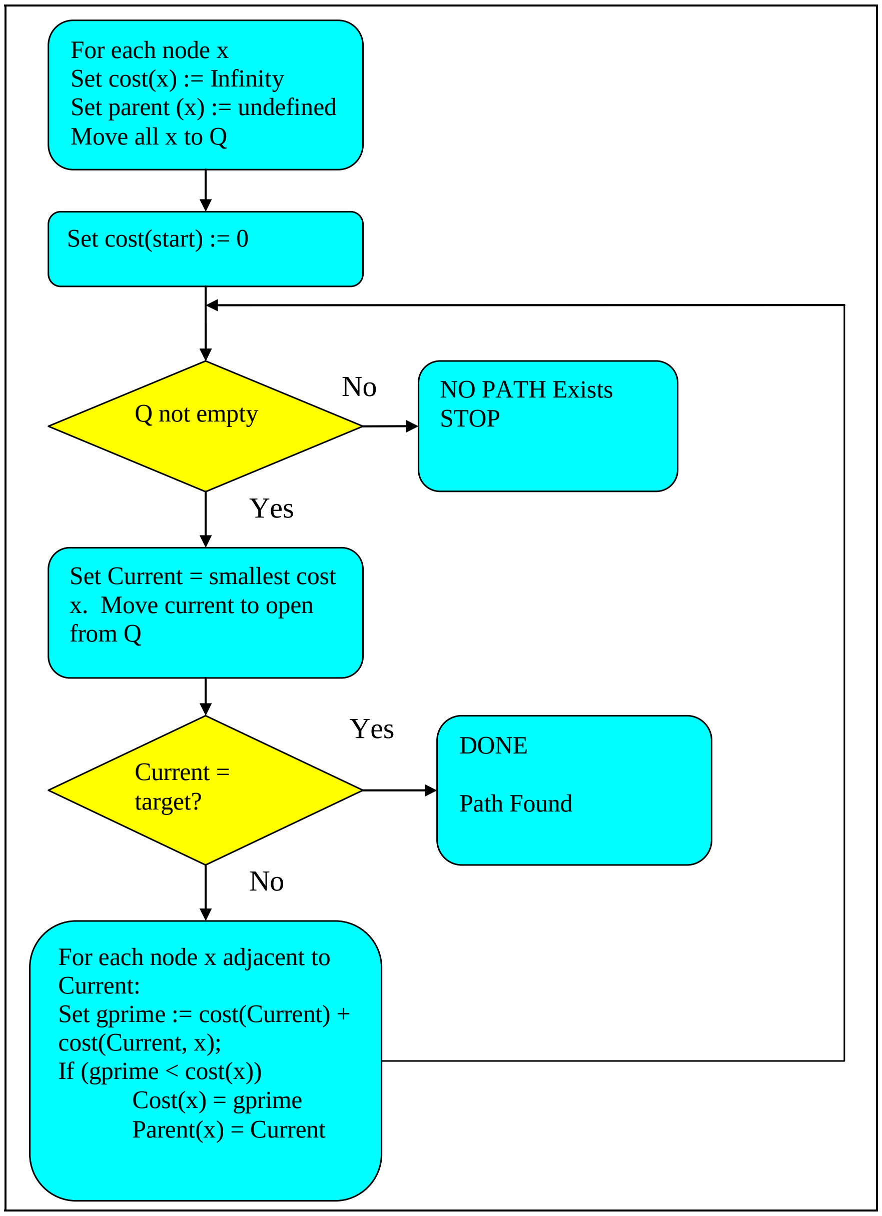


Figure 5. Flow chart of Dijkstra's algorithm.

At this point, we have already checked the costs to each of the nodes adjacent to the start node when we next consider links to all of the (as yet unclosed) nodes adjacent to the closed nodes. Such links will either originate on the start node or the already evaluated node so the net cost will be either c(link) + 0 for links originating on the start node or c(link)+c(1) for links originating on the already evaluated node, which we will call "1". Among all the partial paths just created, one will be the cheapest, and we now have the cost for that node, which we call "2" and can now close. The point is that we can be sure that there is no other path to 2, which could be cheaper, since it would have to have some positive link cost added to getting to some other parent node, which was already more costly to get to than the path currently found. The procedure can be iterated until the target node is reached or all reachable nodes have been searched. If the target node is reached then the chain of parent to child nodes leading to it is a least cost path (there might be more than one).

The algorithm described in the paragraph above does not use the informed search idea, but informed search introduces some complications, as well as efficiencies.

Dijkstra's algorithm takes the brute force path of searching steadily outward. The algorithm first checks all the closest grid elements (for a planar grid or square that would be the four edge-adjacent squares) then the next closest (the four new squares corner-adjacent to the original square) and so on until it reaches the target. One advantage of this method is that each new grid element that is searched has a unique parent square so that once the search pattern reaches the target square, the shortest path or, more precisely, a shortest path (there may be other paths of equal cost) consists of the target square and all of its parents (see figure 6.

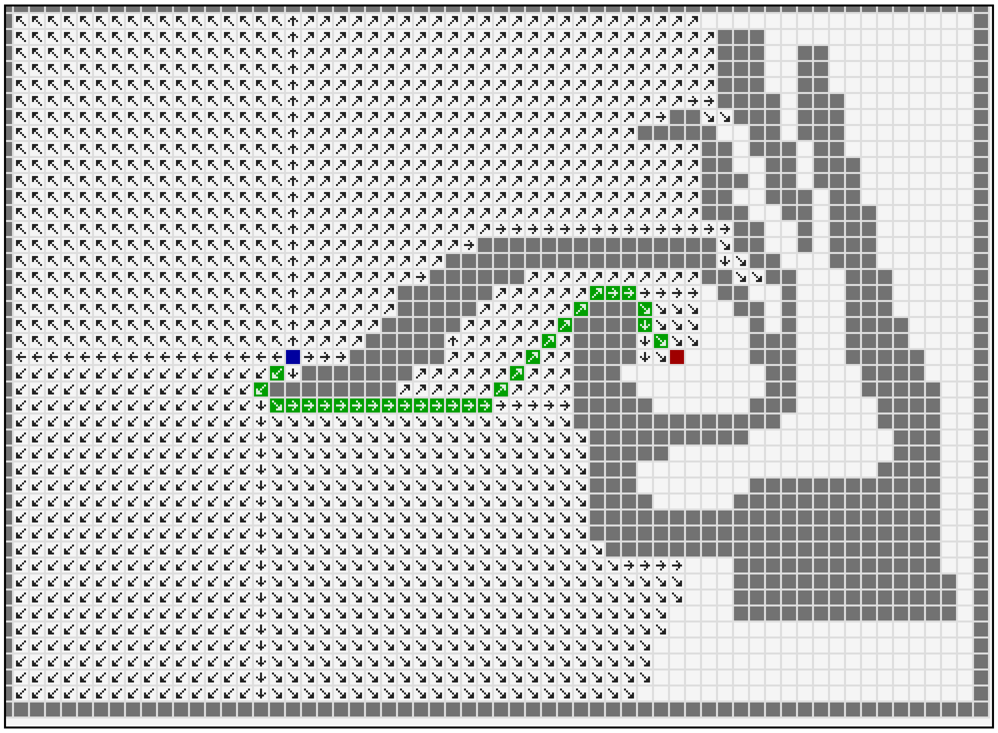


Figure 6. Dijkstra's algorithm.

Dijkstra's algorithm (figure 6) finds the least cost path, but the efficiency is not necessarily ideal. "Blue" is the starting node, "red" is the destination node, and gray squares are blocked. Arrows in squares point to the square (node) and from its parent square. Squares with arrows were tested, and those without were not. The green squares form the path found by the algorithm. This figure was produced using an applet written by Baur (website accessed 2008).

The disadvantage of this algorithm is that all grid elements closer to the origin than the target need to be searched (see figure 6), as do some of those equally close to the origin. Compared to the A* algorithm, it is usually less efficient. Of course most of those grid elements searched won't even be in the right direction as the target (see figure 6 and the figures in Patel [website accessed February 25, 2006]).

6.3.3 Best First Search Algorithm

Another type of algorithm can make use of information like the direction of the target. An example is the so-called Best First Search (BFS) algorithm. When the direction to the target is known, this information can be encoded in a *heuristic* that tells which squares to search first. In this case, it is the square that lies in the direction of the target. Under many circumstances this method is much faster than Dijkstra's algorithm, since if a relatively direct path works, it winds up searching a much more limited set of paths (see figure 5 and figures in Patel [website accessed February 25, 2006]). The most obvious heuristic to use is the distance to the target. The least distance heuristic here serves as the only cost function.

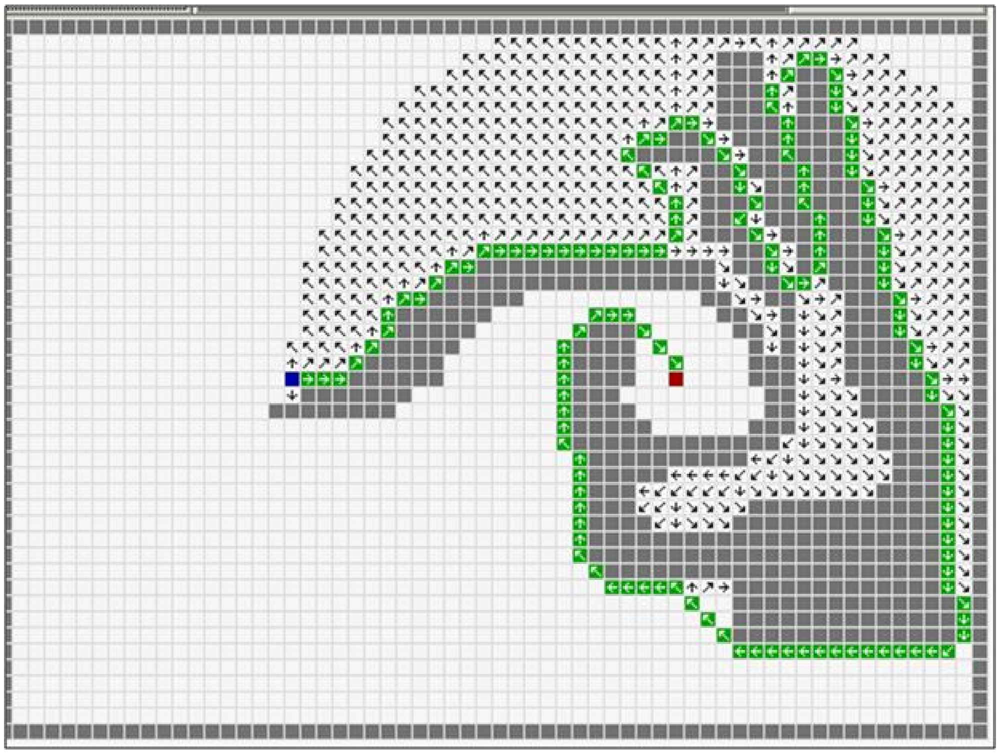


Figure 7. Best First Search algorithm.

The Best First Search algorithm (figure 7) can't resist temptation and finds a seriously non-optimal path. "Blue" is the starting node, "red" is the destination node, and gray squares are blocked. Arrows in squares point to the square (node) and from its parent square. Squares with arrows were tested, and those without were not. The green squares form the path found by the algorithm. Figure 7 was produced using an applet written by Baur (website accessed 2008).

A disadvantage of BFS is that if a concave obstacle is interposed between goal and target, the search path goes to the bottom of the concavity in its attempt to get closer to the target, and the final path will reflect this (see figure 7). Thus, in a complex environment, BFS may find a seriously non-optimal path.

6.3.4 The A* Algorithm

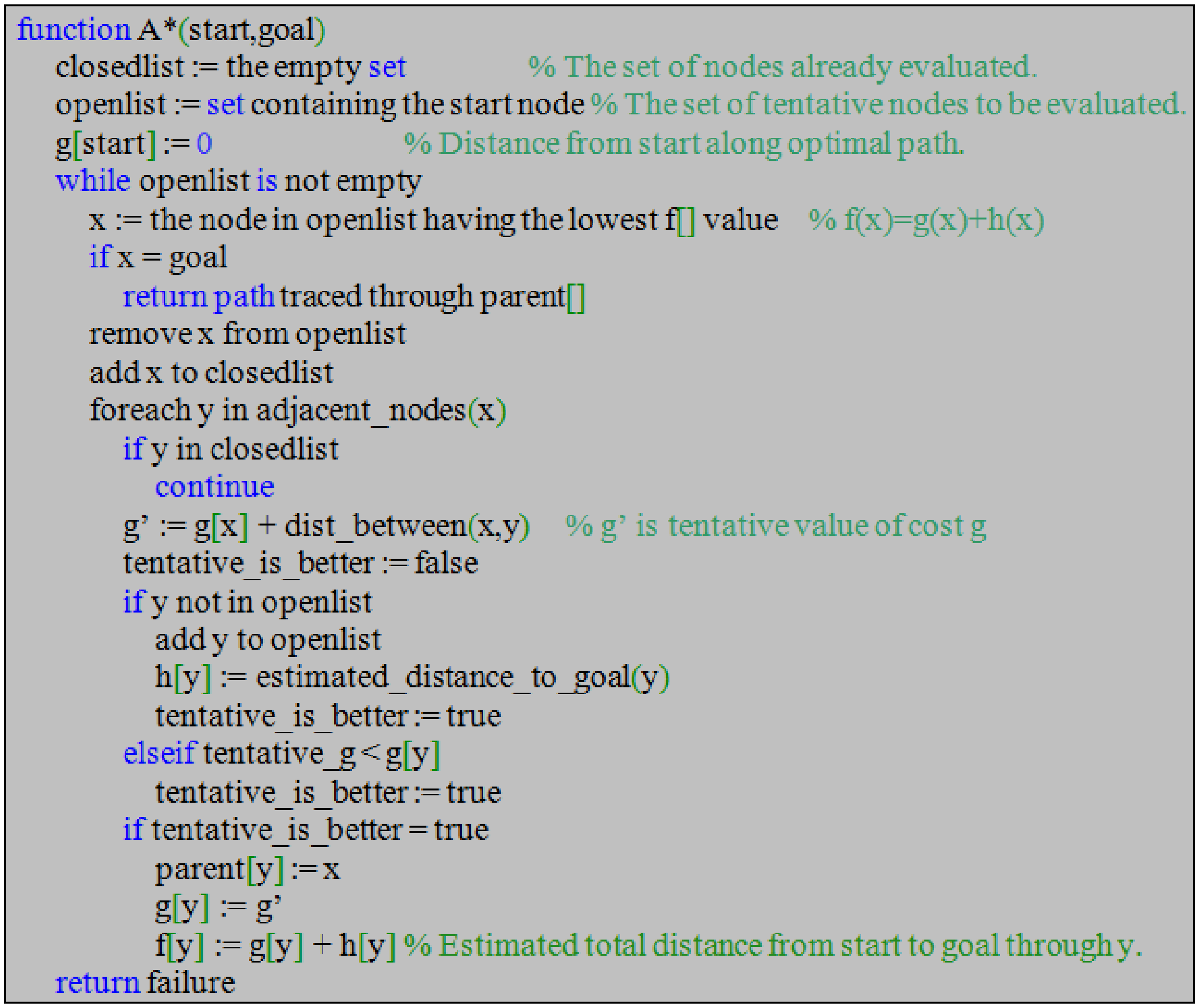
The A* algorithm is a favorite of programmers of three-dimensional computer games. Their problem, which is moving computer-controlled characters from current location to intended destination without falling into a hole, walking through fire, or any of the other innumerable hazards, is strikingly similar to the problem of routing an unmanned vehicle in the real world. The A* algorithm combines the approaches of Dijkstra's algorithm and Best First Search algorithm. To accomplish, this it uses a function $f(x) = g(x) + h(x)$, where $g(x)$ is the cost to get to the node x, and $h(x)$ is the heuristic (ordinary straight line) Euclidean distance from that node to the target location. In effect, the A* algorithm picks the next promising node on its search path by combining $g(x)$, the cost to get to that node from the start, and $h(x)$, the remaining distance to the target.

6.3.4.1 How the A* Algorithm Works

The cost to get from the start to any given point by means of a given path is computed from the cost of traversal from node to node. These costs are computed from the weather risks and other mission risk factors, as discussed earlier.

Like the breadth-first search algorithm, the A* algorithm always searches the lowest f-value direction first, but because its f-value includes the distance to the target, it wastes less time searching directions leading away from the target than a pure Dijkstra's or breadth-first algorithm. Because that f-value contains costs as well, in contrast to the Best first Search algorithm, it doesn't stay distracted by paths that promise but don't deliver.

Figure 8 is an A* algorithm pseudo code, adapted from Wikipedia (website accessed July 30, 2006).



```
function A*(start,goal)
    closedlist := the empty set             % The set of nodes already evaluated.
    openlist := set containing the start node % The set of tentative nodes to be evaluated.
    g[start] := 0                    % Distance from start along optimal path.
    while openlist is not empty
        x := the node in openlist having the lowest f[] value    % f(x)=g(x)+h(x)
        if x = goal
            return path traced through parent[]
        remove x from openlist
        add x to closedlist
        foreach y in adjacent_nodes(x)
            if y in closedlist
                continue
            g' := g[x] + dist_between(x,y)     % g' is tentative value of cost g
            tentative_is_better := false
            if y not in openlist
                add y to openlist
                h[y] := estimated_distance_to_goal(y)
                tentative_is_better := true
            elseif tentative_g < g[y]
                tentative_is_better := true
            if tentative_is_better = true
                parent[y] := x
                g[y] := g'
                f[y] := g[y] + h[y] % Estimated total distance from start to goal through y.
    return failure
```

Figure 8. The A* algorithm pseudo code.

See figure 9 for the A* algorithm flow chart.

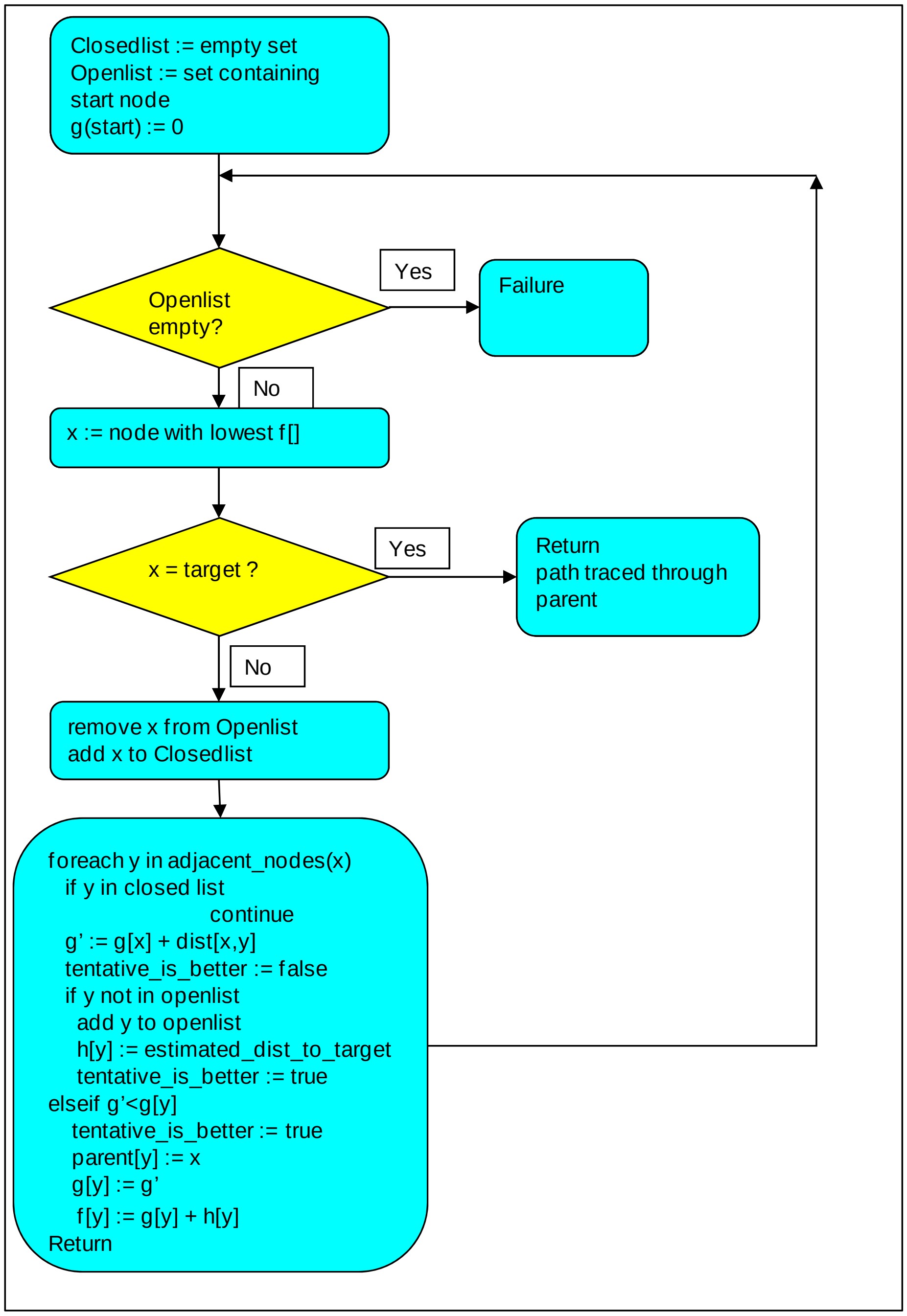


Figure 9. The A* algorithm flow chart.

The A* algorithm itself is fairly simple, and it will always find the best path according to the definition of cost provided to it, assuming that the cost obeys the uniformly-positive condition described. This can result in a very impressive, smart-looking solution considering all factors to a degree impossible for any human analyst. The challenge in implementing a complex path-finding routine is to intelligently define the path cost to accurately direct the algorithm. For naturally scalar costs like weather and flight time this is trivial, but for more complex notions of the "best" path, this can get very difficult.

An example of a feature that does not fit well into this paradigm is the notion of a way point. Determining how to adjust the cost of cells or paths between cells in order to cause the A*algorithm to "try" to include a semi-optional way point in a route without incurring too much cost to get there is not straightforward. This can be accommodated by looking at the paths as a whole, not just segment by segment, and considering "better" those paths that pass close to the way point; however, searching each path for the node closest to the way point every time a path is tested for its cost is very expensive, so using this method efficiently is difficult. A more straightforward approach is to treat each segment as a separate routing problem, but that risks a route that orders the segments sub-optimally. Figure 10 shows the A* algorithm.

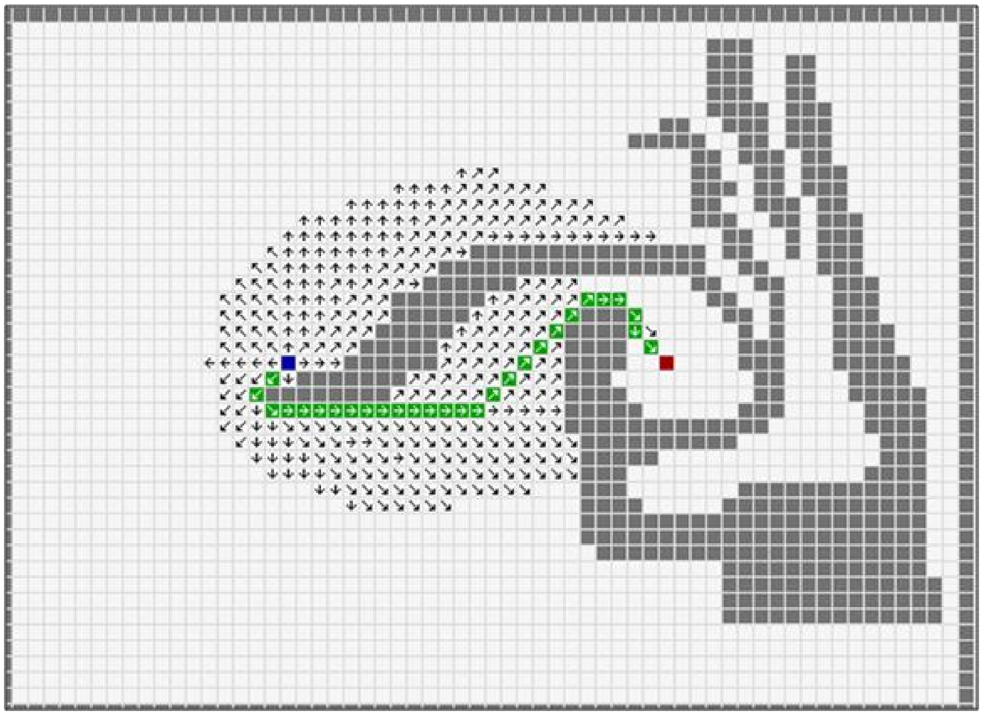


Figure 10. The A* algorithm.

Using both cost and heuristic allows the A* algorithm to find the best path and still be reasonably efficient. "Blue" is the starting node, "red" is the destination node, and gray squares are blocked. Arrows in squares point to the square (node) and from its parent square. Squares with arrows were tested, those without were not. The green squares form the path found by the algorithm. This figure was produced using an applet written by Baur (website accessed 2008).

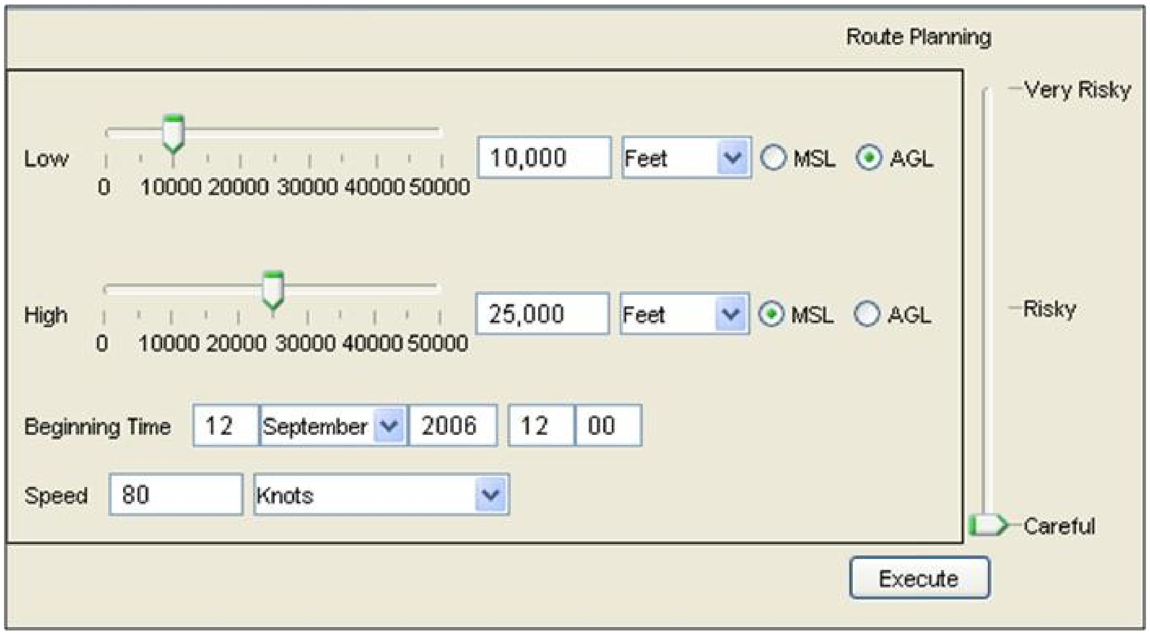
For the example of figure 10, the gain in computational efficiency is obvious. For the three-dimensional case, where in the number of nodes there is a given number of links from the start scales as (2* l+1)^3 rather than (2*l+1)^2 as in the two-dimensional case, even greater gains in efficiency are possible. For example, in a two-dimensional grid, there are 49 nodes within three links from the source, but in three dimensions, there are 243 nodes.

More examples of the A* algorithm in action, many showing intermediate search states, are found in Appendix A.

7. AWRT Examples and Screen Shots

An experimental computer application using real weather data, T-IWEDA weather effects, and real military UAS vulnerabilities has been developed using the A* algorithm. This application has been named the Aviation Weather Routing Tool (AWRT).

The current user interface for the route planning feature in T-IWEDA appears as shown in Figure 11.



Low 10,000 Feet MSL [✓ AGL]

High 25,000 Feet [✓ MSL] AGL

Beginning Time 12 September 2006 12 00

Speed 80 Knots

Figure 11. User interface for T-IWEDA route planning feature.

The user may enter a minimum and maximum altitude, the initial takeoff time, and the speed of the craft, which will be adjusted internally for ascent and descent of the aircraft. This value is interpreted as ground speed and is NOT currently adjusted for winds. The slider on the right allows the user to enter the amount of risk the mission can tolerate.

Below is a T-IWEDA screenshot showing weather impacts and an initial planned flight path (figure 12) that encounters severe “red” and marginal “amber” weather.

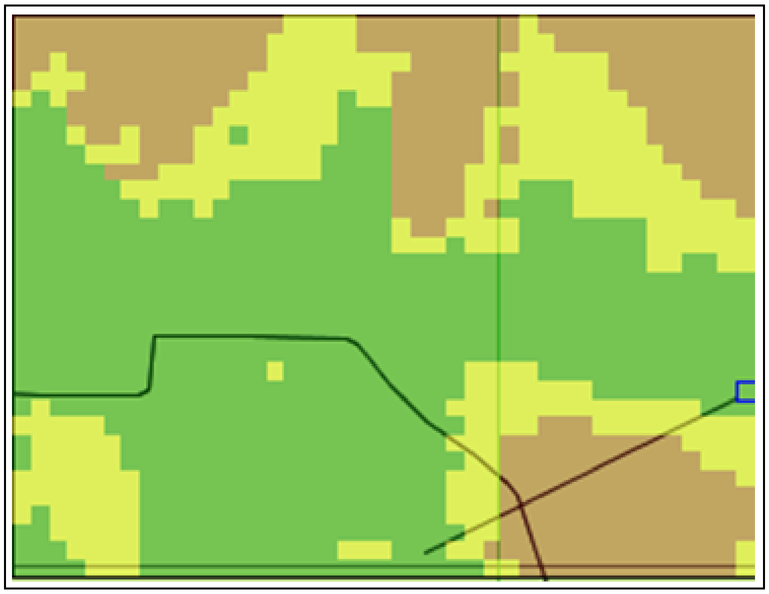


Figure 12. Straight line path for UAS.

After running the route planning algorithm, the following path (figure 13) is generated, which completely skirts the severe and marginal weather, staying almost entirely in the favorable “green” zone, since the routine was run on lowest risk setting.

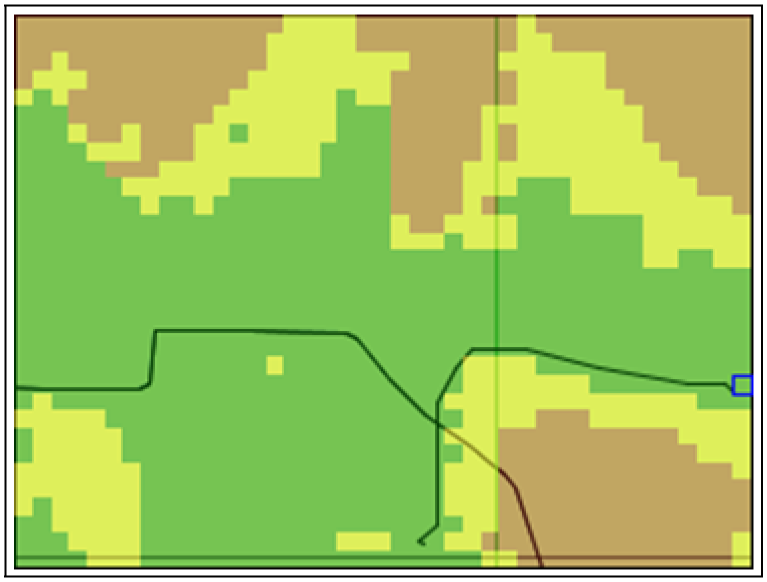


Figure 13. Path optimized for weather effects.

8. Conclusions and Future Directions

Adverse weather impacts significantly impair the effectiveness of our UAS assets. Manual planning to avoid the adverse impacts is complex, labor intensive, and requires extensive training. In order to overcome the disadvantages of manual planning, we have built and demonstrated a mission planning tool for UAS, which finds an optimal route considering weather and weather impacts on the aircraft system. This system is potentially useful for manned aviation mission planning as well.

This tool and its future enhancements can be powerful tools to facilitate and simplify the job of the mission planner. Contemplated real-time versions could also provide crucial data to the mission commander. The work discussed in this report implemented a routing algorithm that minimizes the weather cost of a mission from point-to-point.

The demonstrated system implements a new technology with potential to be extended in more than one direction.

Future enhancements to be considered include addition of new aviation effects, addition of new UAS vehicles to the database, and a major redesign of the graphics.

The first two suggested system enhancements are fairly self-explanatory, but the proposed graphics redesign deserves some explanation. Currently, the graphics are an extremely simple two plus one-dimensional display with a plan view of weather effects and a slice view of a vertical cross-section along the path actually taken. While we expect these views to continue to be useful, it is also felt that a truly three-dimensional fly-through view incorporating both weather and terrain would be valuable. Work on that view is underway.

More fundamentally, UAS are not usually flown in a point-to-point fashion, although that does form an aspect of their operations. UAS can be improved by redesign to incorporate a greater variety of mission scenarios, especially including those most commonly flown in combat conditions. This would require transitioning from minimizing the point-to-point routing cost to minimizing the overall mission risk. Actual UAS missions typically include route patrol in support of convoys or other movements, reconnaissance, surveillance, or operations in support of combat. These missions typically include forms of area patrol in circular or S-shaped loops with periods of very precise mission commander control to track specific targets or other items of interest. The redesigned tool should be useful for mission planning but should also be available during mission execution for assessment of ongoing risks, such as carburetor icing due to water vapor and ambient temperature, as well as for monitoring fuel status in the light of changing wind conditions.

9. References

Baur, Stefan K. Pfadsuche-Applet. http://www.stefan-baur.de/cs.web.mashup.pathfinding.html (accessed 2008).

HQDA. Army Unmanned Aircraft System Operations. *Field Manual Interim* [online] **2006**, No. 3-04.155.

Hart, Peter E.; Nils J. Nilson; Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics* [online] **1968,** Vol. ssc-4, No. 2.

MacGill, James. A* Demonstration. http://www.vision.ee.ethz.ch/~buc/astar/AStar.html (accessed 1999).

Patel, Amit J. Amit's Thoughts on Path-Finding and A-Star. http://theory.stanford.edu/~amitp/GameProgramming/ (accessed February 25, 2006).

Rensselaer Polytechnic Institute. Dijkstra's Algorithm. http://www.ibiblio.org/links/devmodules/graph_networking/compat/page13.html (accessed 1999).

Wikipedia. A* search algorithm. http://en.wikipedia.org/wiki/A-star_search_algorithm#Intuition (accessed July 30, 2006).

Wikipedia. Breadth-first search. http://en.wikipedia.org/wiki/Breadth-first_search (accessed August 8, 2009).

Wikipedia. Dijkstra's algorithm. http://en.wikipedia.org/wiki/Dijkstra's_algorithm (accessed August 18, 2009).

Appendix A. Examples of the A* Algorithm in Action

It is useful to see the A-star (A*) algorithm in action on some simple two-dimensional problems before we look in detail at its application to UAS routing. James MacGill of the University of Leeds has created a demonstration applet that illustrates how the A* algorithm works. In the rest of this section, we will present several screen-shots from the applet with variously difficult obstacles and discuss their content.

Figures A-1 through A-4 demonstrate easy problems.

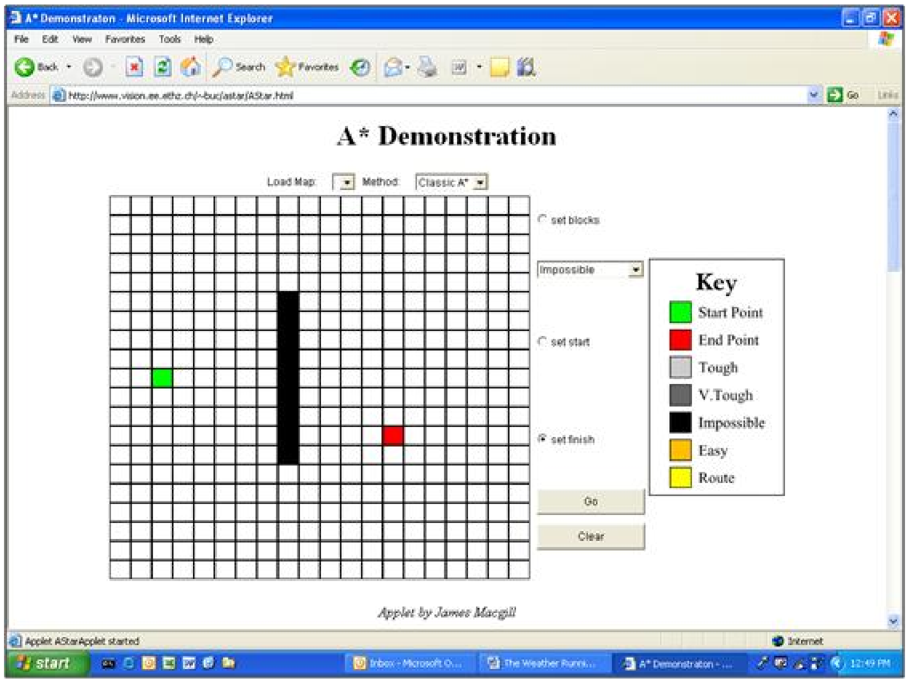
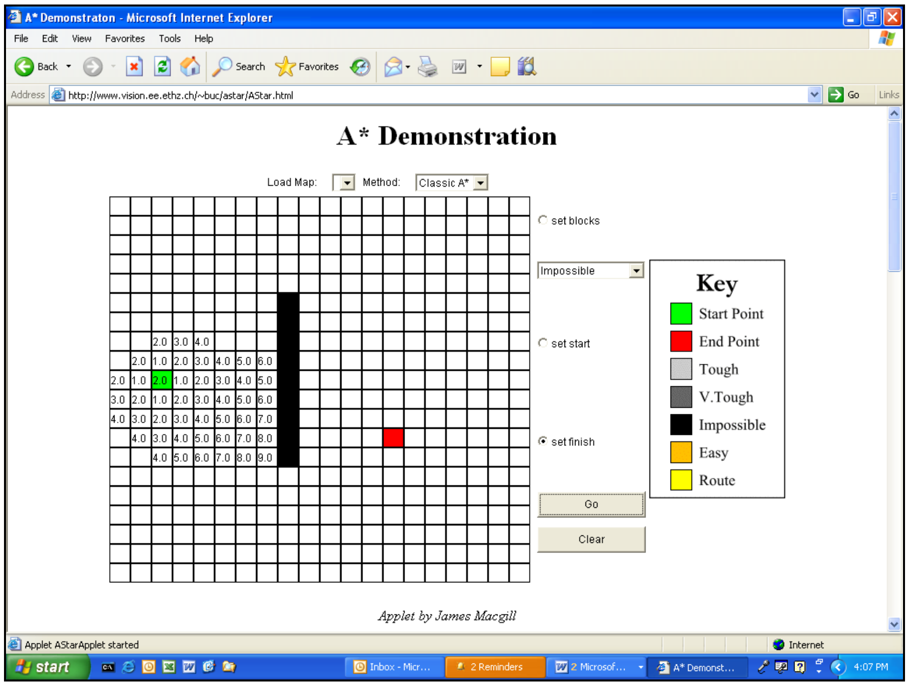


Figure A-1. A simple impenetrable obstacle.

In figure A-1 above, the green square is the origin, the red square is the objective, and the black squares are impenetrable obstacles.

Figures A-2 through A-5 show the solution search at various phases. The numbers in the squares show the computed cost of movement to that square.



A* Demonstration

Load Map: Method: Classic A*

Impossible

[✓ set finish]

Key
🟩 Start Point
🟥 End Point
⬜ Tough
▨ V.Tough
⬛ Impossible
🟧 Easy
🟨 Route

Applet by James Macgill

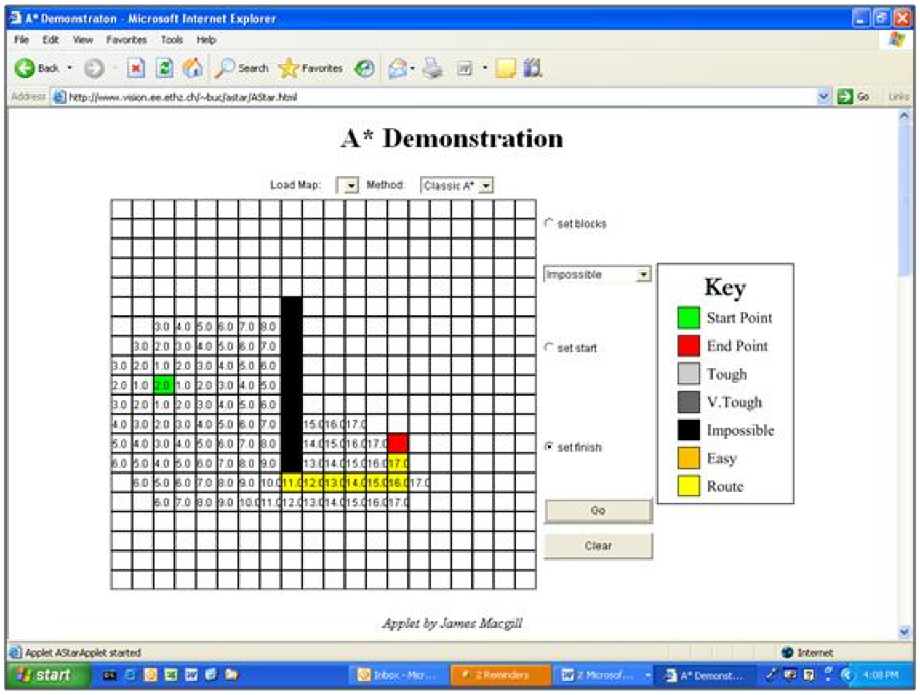
Figure A-2. The initial stages of the A* algorithm search.

In figure A-2 there are a few oddities. The only movements considered are to edge-adjacent squares-squares adjacent at a corner have to be reached through two successive edge-adjacent moves. This is a simplification of the implementation and not a fundamental limitation of the algorithm. Also, the origin square gets a cost-the cost of moving one square away and then back. There are many possible paths to each square, but only the lowest cost one is stored. Since the first value a square gets will be the lowest cost, each square only needs to be looked at once. It wasn't necessary to look at the origin square at all of course, so this is again a peculiarity of the implementation.

Each square of movement here costs one unit, so squares reached in a single move get value 1, those reached in two moves gets value two, and so on. Only the movement costs are shown in the squares here, but remember that in choosing which square to evaluate next, the algorithm looks at the movement cost and the heuristic, the distance from the target. Consider the leftmost element of the top row of evaluated squares. Cost of movement to the square immediately above it would only be three, so why is it not yet evaluated, while the square labeled 9 has been? The square labeled '9' is only seven units from the target (the heuristic doesn't know about obstacles), so f = g + h = 9 + 7 = 16. The other square, by contrast, is 17 units from the target, so its f-value is 3 + 17 = 20.

If we had used the breadth-first Dijkstra's algorithm, it would have checked and scored all squares out to movement cost nine, so the A*algorithm is significantly more efficient.

The next screen shot (figure A-3) shows the search at a slightly later stage.


A* Demonstraton - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.vision.ee.ethz.ch/~buc/astar/AStar.html
Go Links
Go
Clear
Applet AStarApplet started
Internet
start
Inbox - Micr...
2 Reminders
2 Microsof...
A* Demonst...
4:08 PM

A* Demonstration

Load Map: Method: Classic A*

( ) set blocks

Impossible

( ) set start

(•) set finish

Key

🟩	Start Point
🟥	End Point
⬜	Tough
▨	V.Tough
⬛	Impossible
🟧	Easy
🟨	Route

Applet by James Macgill

Figure A-3. Target found and path is being defined.

In the picture above the search has reached the target, at a cost of 18 units. Path (the yellow squares) definition now proceeds by choosing the lowest cost edge-adjacent squares, which are always going downhill. Notice the lowest cost path is not unique. Each monotonically decreasing path from the red square to the yellow square labeled "11.0"' is equivalent in cost.

The last screenshot (figure A-4) from this search shows the final path.

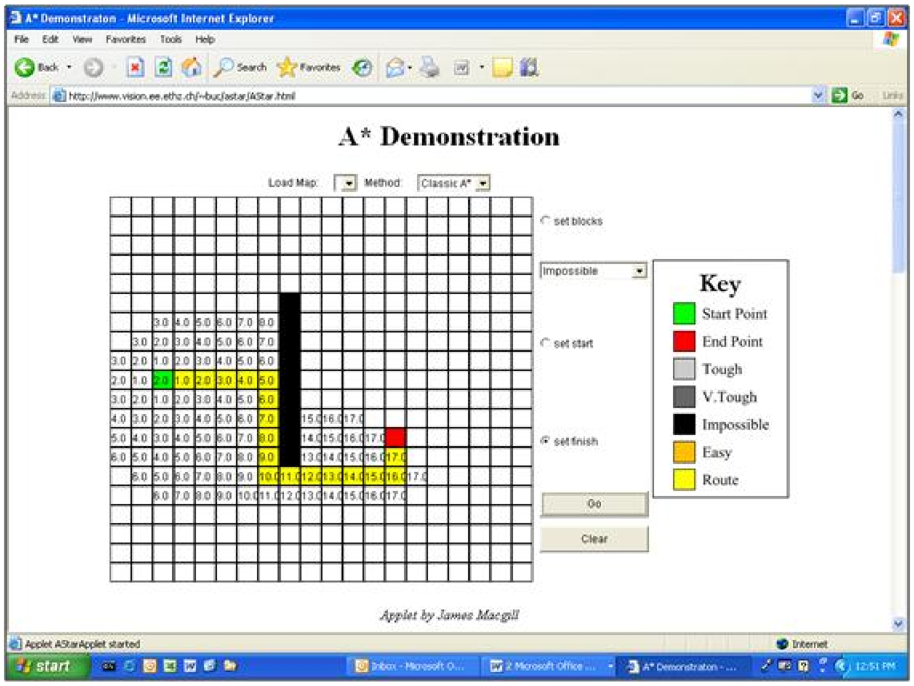


Figure A-4. The shortest (cheapest) path.

Figure A-4 doesn't look like the shortest path until we remember that only edge-adjacent moves are permitted in this demo. Once again, that is an implementation detail, and not a fundamental feature of the algorithm.

Figures A-5 through A-8 demonstrate a complex path.

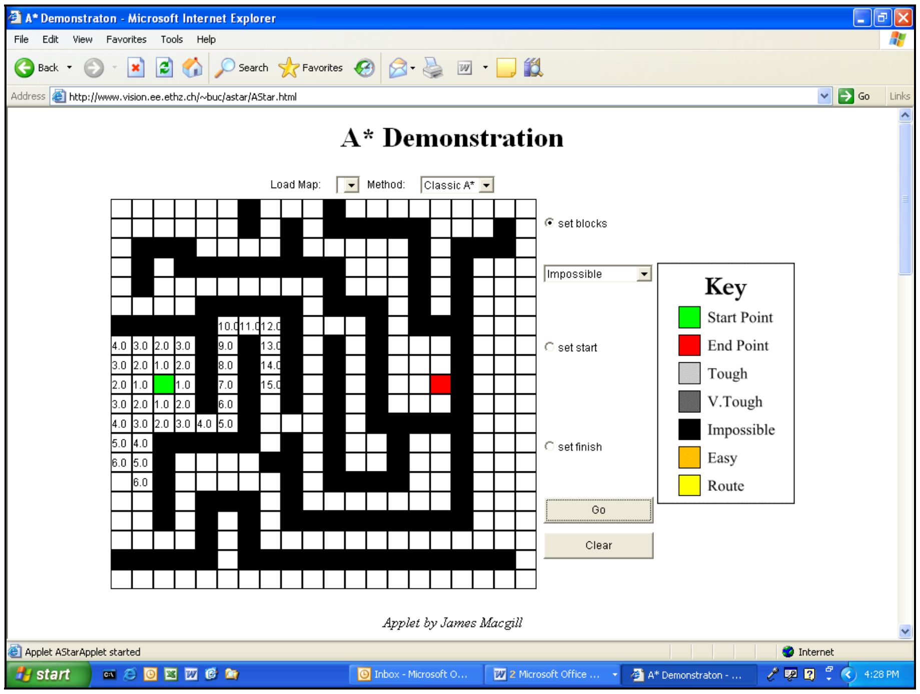
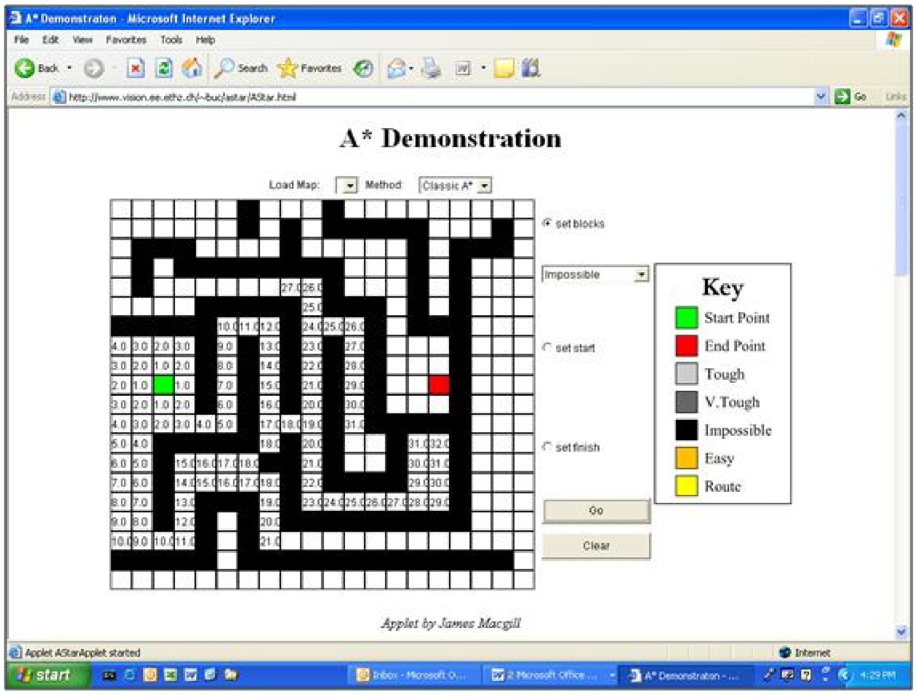


Figure A-5. Tackling a complex path.

The partially solved path in figure A-5 is complex enough that the correct path to the target may not be immediately obvious visually. Here we see the search process at an early stage.

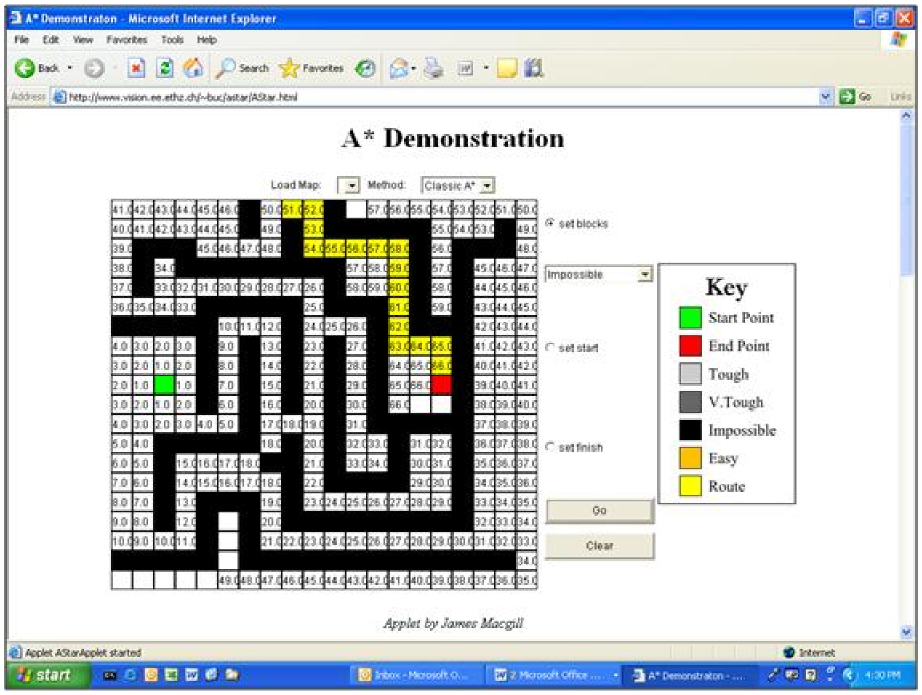
Figure A-6 below takes the search a bit further. Reference figure A-7.



Method: Classic A*

Impossible

Figure A-6. The same search at an intermediate point.

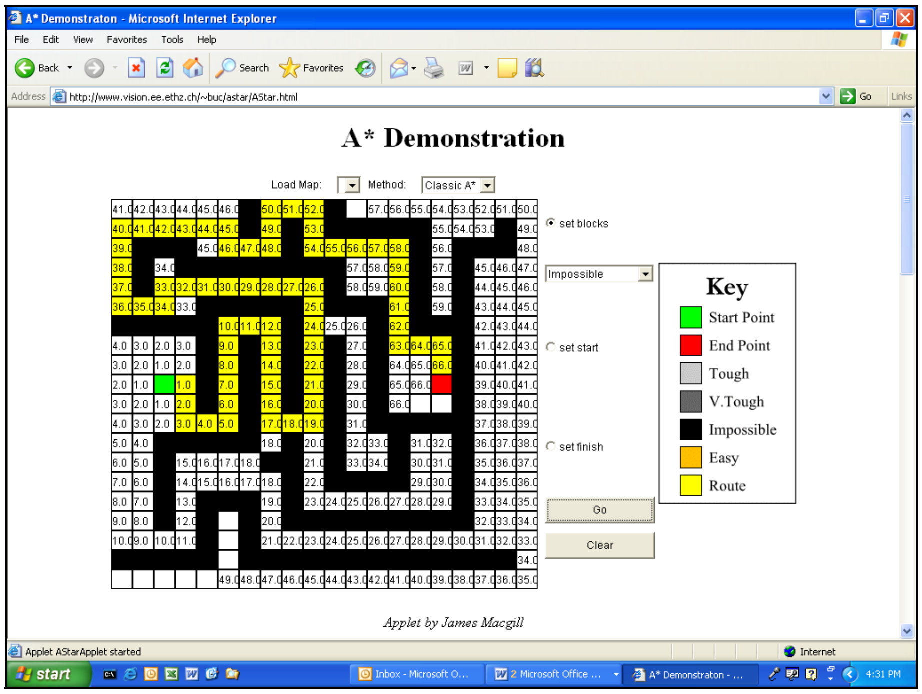


Method: Classic A*

Impossible

Figure A-7. Defining the path.

Figure A-7 shows the trace back of the lowest cost path (yellow colored squares) underway. Recall that when we reach the goal node, the optimal path is constructed by linking each node to its parent until the start is reached. That sequence constitutes the optimal path only in reversed order. This reverse-ordered construction is necessary since each node has a unique parent, but not vice versa. A given node may be the parent of multiple nodes.


A* Demonstraton - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.vision.ee.ethz.ch/~buc/astar/AStar.html
Go Links
Applet AStarApplet started
Internet
start
Inbox - Microsoft O...
2 Microsoft Office ...
A* Demonstraton - ...
4:31 PM

A* Demonstration

Load Map: Method: Classic A*

41.0	42.0	43.0	44.0	45.0	46.0	■	50.0	51.0	52.0	■		57.0	56.0	55.0	54.0	53.0	52.0	51.0	50.0
40.0	41.0	42.0	43.0	44.0	45.0	■	49.0	■	53.0	■	■	■	■	■	55.0	54.0	53.0	■	49.0
39.0	■	■	■	45.0	46.0	47.0	48.0	■	54.0	55.0	56.0	57.0	58.0	■	56.0	■	■	■	48.0
38.0	■	34.0	■	■	■	■	■	■	■	■	57.0	58.0	59.0	■	57.0	■	45.0	46.0	47.0
37.0	■	33.0	32.0	31.0	30.0	29.0	28.0	27.0	26.0	■	58.0	59.0	60.0	■	58.0	■	44.0	45.0	46.0
36.0	35.0	34.0	33.0	■	■	■	■	■	25.0	■	■	■	61.0	■	59.0	■	43.0	44.0	45.0
■	■	■	■	■	10.0	11.0	12.0	■	24.0	25.0	26.0	■	62.0	■	■	■	42.0	43.0	44.0
4.0	3.0	2.0	3.0	■	9.0	■	13.0	■	23.0	■	27.0	■	63.0	64.0	65.0	■	41.0	42.0	43.0
3.0	2.0	1.0	2.0	■	8.0	■	14.0	■	22.0	■	28.0	■	64.0	65.0	66.0	■	40.0	41.0	42.0
2.0	1.0	🟩	1.0	■	7.0	■	15.0	■	21.0	■	29.0	■	65.0	66.0	🟥	■	39.0	40.0	41.0
3.0	2.0	1.0	2.0	■	6.0	■	16.0	■	20.0	■	30.0	■	66.0			■	38.0	39.0	40.0
4.0	3.0	2.0	3.0	4.0	5.0	■	17.0	18.0	19.0	■	31.0	■	■	■	■	■	37.0	38.0	39.0
5.0	4.0	■	■	■	■	■	18.0	■	20.0	■	32.0	33.0	■	31.0	32.0	■	36.0	37.0	38.0
6.0	5.0	■	15.0	16.0	17.0	18.0	■	■	21.0	■	33.0	34.0	■	30.0	31.0	■	35.0	36.0	37.0
7.0	6.0	■	14.0	15.0	16.0	17.0	18.0	■	22.0	■	■	■	■	29.0	30.0	■	34.0	35.0	36.0
8.0	7.0	■	13.0	■	■	■	19.0	■	23.0	24.0	25.0	26.0	27.0	28.0	29.0	■	33.0	34.0	35.0
9.0	8.0	■	12.0	■		■	20.0	■	■	■	■	■	■	■	■	■	32.0	33.0	34.0
10.0	9.0	10.0	11.0	■		■	21.0	22.0	23.0	24.0	25.0	26.0	27.0	28.0	29.0	30.0	31.0	32.0	33.0
■	■	■	■	■		■	■	■	■	■	■	■	■	■	■	■	■	■	34.0
					49.0	48.0	47.0	46.0	45.0	44.0	43.0	42.0	41.0	40.0	39.0	38.0	37.0	36.0	35.0

[✓ set blocks]

Impossible

set start

set finish

Go

Clear

Key

🟩	Start Point
🟥	End Point
⬜	Tough
▩	V.Tough
■	Impossible
🟧	Easy
🟨	Route

Applet by James Macgill

Figure A-8. The shortest path.

Note that in this very complex path problem (figure A-8), the efficiency advantage of the A* algorithm routine is much less pronounced. Nearly the entire search space ended up being tested. Note also that it did find the shortest path solution. Like Dijkstra’s algorithm, the A* algorithm will find a shortest path if it exists, that is, if there is a passable path to the target. For this particular problem, its advantage in efficiency is small, but not zero.

Appendix B. A*Algorithm Code for Route Finding

```
    public GridPath [] optimizeFrom(GridPath aStart, 
arl.tsiweda.ProgressTracker aTracker)
            throws RoutePlanningException
      {
            //Gets the busy message
            String progressMessage = aTracker.getProgressMessage();
            //Adds aStart to the queue of paths to expand
            add(aStart);
            //Start out with aStart as the best path (since it's the only 
path so far)
            GridPath best = aStart;
            GridPath oldBest;
            //Does the cost calculation for the initial path
            aStart.calculateCost();
            int i;
            //The takeoff time
            long time = theSession.theStartTime;
            //Optimizes until the best path is the same as the target point
            for(i = 0; best != null && 
!best.equals(createGridPath(theSession.theEnd, best.theTime)); i++)
            {
                  time = best.theTime;
                  //Updates the progress message once in a while
                  if(i % 1000 == 0)
                        aTracker.setProgressMessage(progressMessage + " (" + 
i + " paths tested)");
                  //Sets the expanded flag
                  best.theExpanded = true;
                  //Expands the node
                  expand(best, aTracker);
                  oldBest = best;
                  //We're done with this path--don't consider it anymore
                  removeBest(oldBest);
                  //Try the next best path
                  best = getBestPath();
            }
            //best is now the actual best path
            if(best == null)
                  throw new DataRangeException("Cannot reach destination 
within available"
                        + " forecast range: " + 
arl.tsiweda.util.IwedaUtils.print(time));
            log.debug("Best(" + i + ")=(" + best.theX + ", " + best.theY + ", 
" + best.theZ + ")");
            aTracker.setProgressMessage(progressMessage);
            //Go back through the parents to get an array representing the 
path
            java.util.List pathList = new java.util.ArrayList();
            while(best != null)
            {
                  pathList.add(best);
                  best = best.theParent;
            }
            java.util.Collections.reverse(pathList);
            GridPath [] ret = (GridPath []) pathList.toArray(new GridPath
[pathList.size()]);
            //Release resources
            clear();
            return ret;
      }
```

List of Symbols, Abbreviations, and Acronyms

A*	A-star
ARL	U.S. Army Research Laboratory
AWRT	Aviation Weather Routing Tool
BED	Battlefield Environment Division
BFS	Best First Search
DoD	Department of Defense
FMI	Field Manual Interim
HQDA	Headquarters, Department of the Army
IMETS	Integrated Meteorological System
IWEDA	Integrated Weather Effects Decision Aid
Met	meteorological
NWP	Numerical Weather Prediction
T-IWEDA	Tri-Service Integrated Weather Effects Decision Aid
TDA	Tactical Decision Aid
UAS	Unmanned Aircraft Systems
UAV	Unmanned Air Vehicle

No. of Copies	**Organization**
1 PDF	ADMNSTR DEFNS TECHL INFO CTR DTIC OCP 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
3 HCs	US ARMY RSRCH LAB ATTN RDRL CIM P TECHL PUB ATTN RDRL CIM L TECHL LIB ATTN IMNE ALC HRR MAIL & RECORDS MGMT 2800 POWDER MILL ROAD ADELPHI MD 20783-1197
1 CD	US ARMY RSRCH LAB ATTN RDRL CIM G TECHL LIB BUILDING 4600 APG MD 21005-5066
5 CDs	US ARMY RSRCH LAB RDRL CIE D EDWARD M MEASURE BLDG 1622 WSMR NM 88011
5 CDs	US ARMY RSRCH LAB RDRL CIE M DAVID KNAPP BLDG 1622 WSMR NM 88011
5 CDs	US ARMY RSRCH LAB RDRL CIE M TERRY JAMESON BLDG 1622 WSMR NM 88011
5 CDs	PHYSICAL SCIENCE LABORATORY RDRL CIE D ANDREW BUTLER NEW MEXICO STATE UNIVERSITY P.O. BOX 30002 LAS CRUCES, NM 88003-8002
Total:	25 (1 PDF, 21 CDS, 3 HCs)